ALAGAPPA UNIVERSITY

[Accredited with 'A+' Grade by NAAC (CGPA:3.64) in the Third Cycle and Graded as Category–I University by MHRD-UGC]

(A State University Established by the Government of Tamil Nadu)

KARAIKUDI – 630 003

Directorate of Distance Education

Mobile Application Development

Author: Dr.P.Thiyagarajan

Assistant Professor & Head In-charge

Department of Computer Science

Central University of Tamil Nadu

Thiruvarur – 610 005

Email: thiyagu@cutn.ac.in

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu,
Thiruvarur – 610 005

CC XIII – Mobile Application Development

UNIT I

Introduction- The Mobile Ecosystem: Operators - Networks - Devices - Platforms - Operating Systems - Application Frameworks - Applications - Services

UNIT II

Mobile Devices Profiles - Options for development - Categories of Mobile Applications: SMS - Mobile Websites - Mobile Web Widgets - Native Applications - Games - Utility Apps – Location, Based Services(LBS) Apps - Informative Apps - Enterprise Apps

UNIT III

Mobile Information Architecture: Introduction - Sitemaps - Click Streams - Wireframes - Prototyping - Architecture for Different Devices. Mobile Design: Interpreting Design – Elements of Mobile Design - Mobile Design Tools - Designing for Different Device/ Screens

UNIT IV

J2ME Overview -J2ME Architecture and Development Environment- Small Computing Device Requirements, Run-Time Environment, MIDlet Programming, Java Language for J2ME, J2ME SDK, J2ME Wireless Toolkit

UNIT V

Case Study: Google Android: Introduction - Android Development Environment- evelopment Framework- SDK, Eclipse - Emulator, Android AVD - Project Framework. Apple IOS - RIM Blackberry - Samsung Bada - Nokia Symbian - Microsoft Windows Phone.

TEXT BOOKS:

1. Mobile Design and Development by Brian Fling, O'Reilly Media, Inc 2009
2. J2ME: The Complete Reference, James Keogh, Tata McGrawHill 2003

REFERENCE BOOKS:

1. Smart Phone and Next-Generation Mobile Computing by Pei Zheng and Lionel Ni, Elseveir 2006
2. Beginning Android by Mark L. Murphy , Apress 2009

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005

CONTENTS

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

UNIT 1 MOBILE ECOSYSTEM

1.1 Contents of the unit

1.2. Introduction

Mobile Ecosystem is collection of multiple operators, networks, devices, operating system, applications, process and services by companies.

- Mobile devices: Mobile phones, Tablets, Phablets, etc.
- Software: Operating system, Network protocols, development tools, testing tools, etc.
- Companies: Device manufacturers, carrier, Apps stores, development and testing companies. Etc.
- Data: SMS, contacts, bank transactions, etc.
- Mobile manufacturers: Samsung, Blackberry, Sony, Nokia, Motorola, Windows Phone, etc.
- Operating system: IOS, Android, Blackberry OS, Symbian, etc.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

1.3. Objectives

- To study the history from telephone to smart phone.
- To understand the basic concepts of mobile system.
- To acquire more knowledge in mobile Ecosystem.
- Be familiar with network concepts related to mobile Ecosystem.
- To study different mobile operating system.
- To gain knowledge about different mobile platforms and application development.

1.4 Introduction to Mobile system

A mobile phone is a wireless handheld device that allows users to make and receive calls and to send text messages, among other features. The earliest generation of mobile phones could only make and receive calls. Today's mobile phones, however, are packed with many additional features, such as web browsers, games, cameras, video players and even navigational systems.

A mobile phone may also be known as a cellular phone or simply a cell phone. The original Mobile Devices and C The original Mobile Devices and C ommunication was mmunication was designed for voice applications (2G).

When the first mobile phones were introduced, their only function was to make and receive calls, and they were so bulky it was impossible to carry them in a pocket.

Later, mobile phones belonging to the Global System for Mobile Communications (GSM) network became capable of sending and receiving text messages. As these devices evolved, they became smaller and more features were added, such as multimedia messaging service (MMS), which allowed users to send and receive images. Most of these MMS-capable devices were also equipped with cameras, which allowed users to capture photos, add captions, and send them to friends and relatives who also had MMS-capable phones.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

1.4.1 The Mobile Ecosystem

The smart phones become our inseparable companions today. It is become part of human life. A smart phone is a mobile phone that can do more than other phones. They work as a computer but are mobile devices small enough to fit in a user's hand.

Smart phones include the following uses:

- Sending and receiving emails, text, photographs and multimedia messages
- Registering contacts, calculator, currency, alarm, etc. functions
- Browsing the Internet using a mobile browser
- Playing games and Video chat
- point of sale terminal when paying for goods or services
- barcode scanning, creating high quality photographs or video
- Determining user's exact location utilizing GPS (global positioning system) satellites

Upto 1980, we used telephone to exchange voice information over telephone lines. The traditional telephone is shown in figure 1-1



Figure 1-1 Traditional telephone

A cellphone is simply a telephone that doesn't need a landline connection. It enables the user to make and receive phone calls. Some cellphones also offer text messaging.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

A smartphone has more advanced features, including web browsing, software applications and a mobile OS. In turn, a smartphone also offers capabilities such as support for biometrics, video chatting, digital assistants and much more.



Figure 1-2 A modern mobile phone – martphone

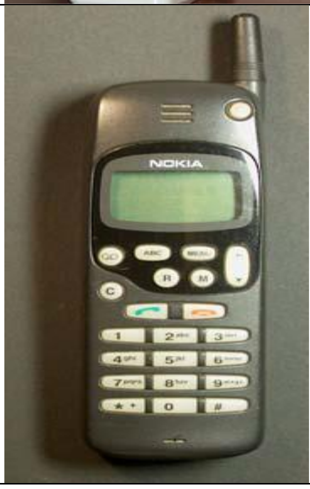
The following are some of the other key features of a smartphone:

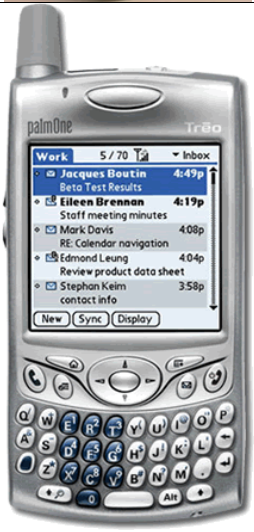
- internet connectivity;
- a mobile browser;
- the ability to sync more than one email account to a device;
- embedded memrory;
- a hardware or software-based QWERTY keyboard;
- wireless synchronization with other devices, such as laptop or desktop computers;
- the ability to download applications and run them independently;
- support for third-party applications;

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

- the ability to run multiple applications simultaneously;
- touchscreen;
- Wi-Fi;
- a digital camera, typically with video capability;
- gaming;
- unified messaging; and
- GPS -- global positioning system.

The Evolution of Devices

Era	Description	Picture
The Brick Era	The first era I call the Brick Era (1973–1988). basically a corded receiver connected to a portable radio the size (and weight) of a car battery.	
The Candy Bar Era	The second era, the Candy Bar Era (1988–1998), represented one of the more significant leaps in mobile technology.	

The Feature Phone Era	The third era, the Feature Phone Era (1998–2008), wasn't nearly as radical a technological leap as the leap from the Brick Era to the Candy Bar Era, but it was an important evolution nonetheless.	
The Smartphone Era	The Smartphone Era occurred at the same time as the third and fifth eras and spans from around 2002 to the present. sending an SMS, taking a picture, and accessing the mobile web, larger screen size, a QWERTY keyboard or stylus for input, and Wi-Fi or another form of high-speed wireless connectivity.	
The Touch Era	crammed into smaller and smaller packages. Mobile devices got smarter by learning from desktop computing, truly becoming personal computers, but people weren't interested.	

The Internet is actually a complex ecosystem made up of many parts that must all work together seamlessly. When we enter a URL into a web browser, we don't think about everything that has to happen to see a web page. When we send an email, we don't care about all the servers,

switches, and software that separate we from our recipient. Everything we do on the Internet happens in fractions of a second. And we have the perception that all of this happens for free.

If we talk to people unfamiliar with mobile, we might find that they quickly assume that the mobile ecosystem is exactly like the Internet, and that all the same rules apply. This couldn't be further from the truth. Mobile is an entirely unique ecosystem and, like the Internet, it is made up of many different parts that must all work seamlessly together. However, with mobile technology, the parts are different, and because we can use mobile devices to access the Internet, that means that not only do we need to understand the facets of the Internet, but we also need to understand the mobile ecosystem.

To put it another way, think of the Internet as a great cloud in the sky. When we want to pull something from it, we use a tool, like a piece of software or device, to interact with it. This can include mobile devices, which we tend to think of as tools. Although this image is partially correct, it's still missing a big piece of the puzzle. To continue the analogy, if the Internet is a cloud, then the mobile ecosystem would be the atmosphere, made up of many clouds, keeping the clouds from drifting off into space; the Internet is just one of these clouds, albeit a very large one.

In case that isn't confusing enough, people in mobile tend to refer to everything related to mobile as "mobile." This chapter looks at some of the clouds in the sky and how each part plays into the ecosystem as a whole. It also looks at how we can get started with mobile.

Think of the mobile ecosystem instead as a system of layers, as shown in Figure 1-3. Each layer is reliant on the others to create a seamless, end-to-end experience. Although not every piece of the puzzle is included in every mobile product and service, for the majority of the time, they seem to add complexity to our work, regardless of whether we expressly put them there.

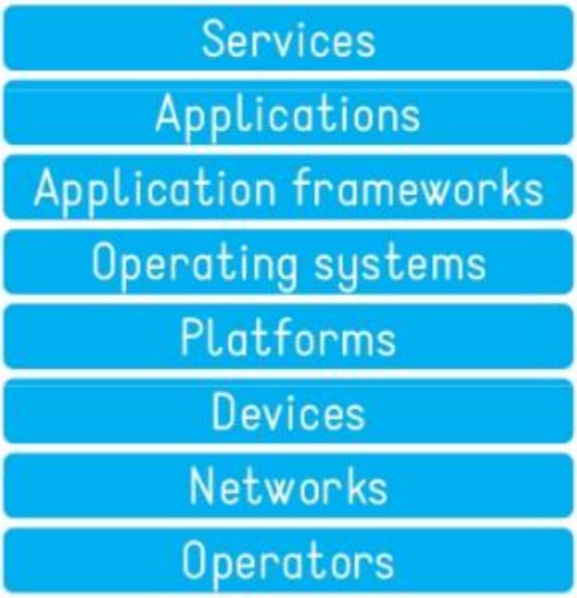


Figure 1-3. The layers of the mobile ecosystem

The following sections expand on each of these layers and the roles they play in the mobile ecosystem.

1.4.2 Operators

The base layer in the mobile ecosystem is the operator. Operators go by many names, depending on what part of the world we happen to be in or who we are talking to. Operators can be referred to as Mobile Network Operators (MNOs); mobile service providers, wireless carriers, or simply carriers; mobile phone operators; or cellular companies. In the mobile community, we officially refer to them as operators, though in the United States, there is a tendency to call them carriers.

Operators are what essentially make the entire mobile ecosystem work. They are the gatekeepers to the kingdom. They install cellular towers, operate the cellular network, make services (such as the Internet) available for mobile subscribers, and they often maintain relationships with the subscribers, handling billing and support, and offering subsidized device sales and a network of retail stores.

The operator's role in the ecosystem is to create and maintain a specific set of wireless services over a reliable cellular network. That's it. However, to grow the mobile market over the past decade, the operator has been required to take a greater role in the mobile ecosystem, doing much more than just managing the network. For example, they have had to

establish trust with subscribers to handle the billing relationship and to offer devices, content, and services that often compete with their partners, who are people like us and who want to create content and services for mobile devices.

Unless we work for an operator, we likely curse their names, at least behind their backs. The operator is viewed as an unfortunate necessity in the mobile world. Often the mobile startups and companies that succeed are the ones with the best "carrier relations man," or person with the best relationship to the operators.

Table 1-4 lists the rank, markets, technologies used, and subscriber numbers for the world's largest operators.

Table 1-4. World's largest mobile operators

Rank	Operator	Markets	Technology	Subscribers (in millions)
1	China Mobile	China (including Hong Kong) and Pakistan	GSM, GPRS, EDGE, TD-SCDMA	436.12
2	Vodafone	United Kingdom, Germany, Italy, France, Spain, Romania, Greece, Portugal, Netherlands, Czech Republic, Hungary, Ireland, Albania, Malta, Northern Cyprus, Faroe Islands, India, United States, South Africa, Australia, New Zealand, Turkey, Egypt, Ghana, Fiji, Lesotho, and Mozambique	GSM, GPRS, EDGE, UMTS, HSDPA	260.5
3	Telefónica	Spain, Argentina, Brazil, Chile, Colombia, Ecuador, El Salvador, Guatemala, Mexico, Nicaragua, Panama, Peru, Uruguay, Venezuela, Ireland, Germany, United Kingdom, Czech	CDMA, CDMA2000 1x, EV-DO, GSM, GPRS, EDGE, UMTS, HSDPA	188.9

		Republic, Morocco, and Slovakia		
4	América Móvil	United States, Argentina, Chile, Colombia, Paraguay, Uruguay, Mexico, Puerto Rico, Ecuador, Jamaica, Peru, Brazil, Dominican Republic, Guatemala, Honduras, Nicaragua, Ecuador, and El Salvador	CDMA, CDMA2000 1x, EV-DO, GSM, GPRS, EDGE, UMTS, HSDPA	172.5
5	Telenor	Norway, Sweden, Denmark, Hungary, Montenegro, Serbia, Russia, Ukraine, Thailand, Bangladesh, Pakistan, and Malaysia	GSM, GPRS, EDGE, UMTS, HSDPA	143.0
6	China Unicom	China	GSM, GPRS	127.6
7	T-Mobile	Germany, United States, United Kingdom, Poland, Czech Republic, Netherlands, Hungary, Austria, Croatia, Slovakia, Macedonia, Montenegro, Puerto Rico, and U.S. Virgin Islands	GSM, GPRS, EDGE, UMTS, HSDPA	126.6
8	TeliaSonera	Norway, Sweden, Denmark, Finland, Estonia, Latvia, Lithuania, Spain, and Central Asia	GSM, GPRS, EDGE, UMTS, HSDPA	115.0
9	Orange	France, United Kingdom, Switzerland, Poland, Spain, Romania, Moldova, Slovakia, Belgium, Liechtenstein, Israel, Egypt, Ivory Coast, Jordan, Cameroon, Botswana, Madagascar, Mali, Senegal, Mauritius, Réunion, Martinique, French Guiana, Saint Kitts	GSM, GPRS, EDGE, UMTS, HSDPA	111.8

		and Nevis, Dominica, and Dominican Republic		
10	MTS	Russia, Ukraine, Belarus, Uzbekistan, Turkmenistan, and Armenia	GSM, GPRS, EDGE, UMTS	91.7
11	MTN	Group Afghanistan, Benin, Botswana, Cameroon, Republic of Congo, Côte d'Ivoire, Cyprus, Ghana, Guinea Bissau, Republic of Guinea, Iran, Liberia, Nigeria, Rwanda, South Africa, Sudan, Swaziland, Syria, Uganda, Yemen, and Zambia	GSM, GPRS, EDGE, UMTS, HSDPA, HSUPA	80.7
12	AT&T	United States, Puerto Rico, and U.S. Virgin Islands	GSM, GPRS, EDGE, UMTS, HSDPA	74.9
13	Bharti Airtel	India, Seychelles, Jersey, Guernsey, and Sri Lanka	GSM, GPRS, EDGE	72.0
14	Verizon Wireless	United States	CDMA2000 1x, EV-DO	70.8
15	SingTel	Singapore, Australia, India, Indonesia, Thailand, Philippines, Bangladesh, and Pakistan	GSM, UMTS, HSDPA	70.7
16	Telecom	Italy, Brazil, San Marino, and Vatican City	GSM, GPRS, EDGE, UMTS, HSDPA	70.6
17	Etisalat	Afghanistan, Benin, Burkina Faso, Central African Republic, the Ivory Coast, Egypt, Gabon, Indonesia, Niger, Nigeria, Pakistan, Saudi Arabia, Sudan, Tanzania,	GSM, GPRS, EDGE, UMTS, HSDPA	63.0

		Togo, and United Arab Emirates		
18	Orascom	Algeria, Bangladesh, Egypt, Pakistan, Tunisia, and Zimbabwe	GSM, GPRS, EDGE	62.9
19	VimpelCom	Russia, Kazakhstan, Ukraine, Uzbekistan, Tajikistan, Georgia, Armenia, Vietnam, and Cambodia	GSM, GPRS, UMTS	57. 8
20	NTT docomo	Japan and Bangladesh	GSM, GPRS, PDC FOMA, HSDPA	53.5

Although most operators are interested in innovation in the wireless marketplace, they have been known to strangle startups with impossible requirements, such as supporting too many devices or seemingly ridiculous certification processes and bad pricing models.

The days of impossible requirements are changing, however. Today's mobile startups have learned the lessons of the companies that came before them that tried to dance with the devil and lost everything. Many look to the successes of Web 2.0–era startups that were able to start with little infrastructure and quickly grow successful businesses. These startups figured out how to duplicate the phenomenon of mobile, bypassing the operators completely (something this book tells we how to do).

We can compare operators to Big Oil. They both have this thing they know everyone wants, and therefore they can make a lot of money from it. They know they have a limited amount of time to make it. With oil, it is the depleted resources and competition of green energy sources; in wireless, it's the growth of competing wireless technologies, such as Wi-Fi, WiMAX, ultra-wide broadband, and whitespace frequencies.

As competing technologies become more mature, they can't charge as much as they did when they first came out. As consumer options in the market mature, both the oil industry and operators must realize that they can't continue to monopolize their markets. They must realize that they don't control their industries; they are only a player in them. Unfortunately, in the meantime, both of these industries will continue to force us to pay an artificially inflated cost to play.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

1.4.3 Networks

A cellular network or mobile network is a communication network where the last link is wireless. The network is distributed over land areas called "cells", each served by at least one fixed-location transceiver, but more normally, three cell sites or base transceiver stations. These base stations provide the cell with the network coverage which can be used for transmission of voice, data, and other types of content. A cell typically uses a different set of frequencies from neighbouring cells, to avoid interference and provide guaranteed service quality within each cell.

When joined together, these cells provide radio coverage over a wide geographic area. This enables numerous portable transceivers (e.g., mobile phones, tablets and laptops equipped with mobile broadband modems, pagers, etc.) to communicate with each other and with fixed transceivers and telephones anywhere in the network, via base stations, even if some of the transceivers are moving through more than one cell during transmission.

Operators operate wireless networks. Remember that cellular technology is just a radio that receives a signal from an antenna. The type of radio and antenna determines the capability of the network and the services we can enable on it.

The vast majority of networks around the world use the GSM standard (see Table for an explanation of these acronyms), using GPRS or GPRS EDGE for 2G data and UMTS or HSDPA for 3G. We also have CDMA (Code Division Multiple Access) and its 2.5G hybrid CDMA2000, which offers greater coverage than its more widely adopted rival. So in places like the United States or China, where people are more spread out, CDMA is a great technology. It uses fewer towers, giving subscribers fewer options as they roam networks.

The first commercial cellular network, the 1G generation, was launched in Japan by Nippon Telegraph and Telephone (NTT) in 1979, initially in the metropolitan area of Tokyo. Within five years, the NTT network had been expanded to cover the whole population of Japan and became the first nationwide 1G network. It was an analog wireless network.

Table 1-5. GSM mobile network evolutions

2G	**Second generation of mobile phone standards and technology**	**Theoretical max data speed**
GSM	Global System for Mobile communications	12.2 KB/sec
GPRS	General Packet Radio Service	Max 60 KB/sec
EDGE	Enhanced Data rates for GSM Evolution	59.2 KB/sec
HSCSD	High-Speed Circuit-Switched Data	57.6 KB/sec

3G	**Third generation of mobile phone standards and technology**	**Theoretical max data speed**
W-CDMA	Wideband Code Division Multiple Access	14.4 MB/sec
UMTS	Universal Mobile Telecommunications System	3.6 MB/sec
UMTS-TDD	UMTS +Time Division Duplexing	16 MB/sec
TD-CDMA	Time Divided Code Division Multiple Access	16 MB/sec
HSPA	High-Speed Packet Access	14.4 MB/sec
HSDPA	High-Speed Downlink Packet Access	14.4 MB/sec
HSUPA	High-Speed Uplink Packet Access	5.76 MB/sec

Like all things in mobile, we like to merge a lot of technology into overly simplistic terms, which tends to create a lot of confusion. So when we say 3G, for example, we often aren't talking about just the capabilities of the network, but the devices that run on it.

Although the core technology that empowers voice communication has stayed relatively the same, network generations are most often used to describe the data speeds the network is capable of delivering.

1.5 Devices

What we call phones, the mobile industry calls handsets or terminals. These are terms that I think are becoming outdated with the emergence of wireless devices that rely on operator networks, but do not make phone calls. The number of these “other” devices is a small piece of the overall pie right now, but it’s growing rapidly.

Let’s focus on the biggest slice of the device pie—mobile phones. As of 2008, there are about 3.6 billion mobile phones currently in use around the world; just more than half the planet’s population has a mobile phone (see Figure 2-2).

Most of these devices are feature phones, making up the majority of the marketplace. Smartphones make up a small sliver of worldwide market share and maintain a healthy percentage in the United States and the European Union; smartphone market share is growing with the introduction of the iPhone and devices based on the Android platform. As next-generation devices become a reality, the distinction between feature phones and smartphones will go away.

In the next few years, feature phones will largely be located in emerging and developing markets. Figure 1-4 shows a breakdown of devices.

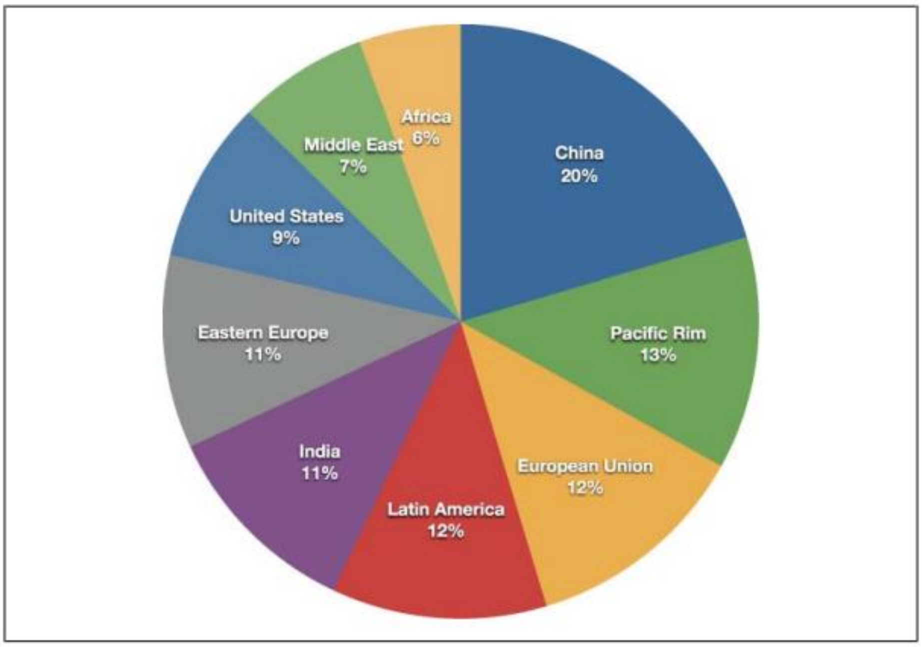


Figure 1-4. Mobile devices around the world

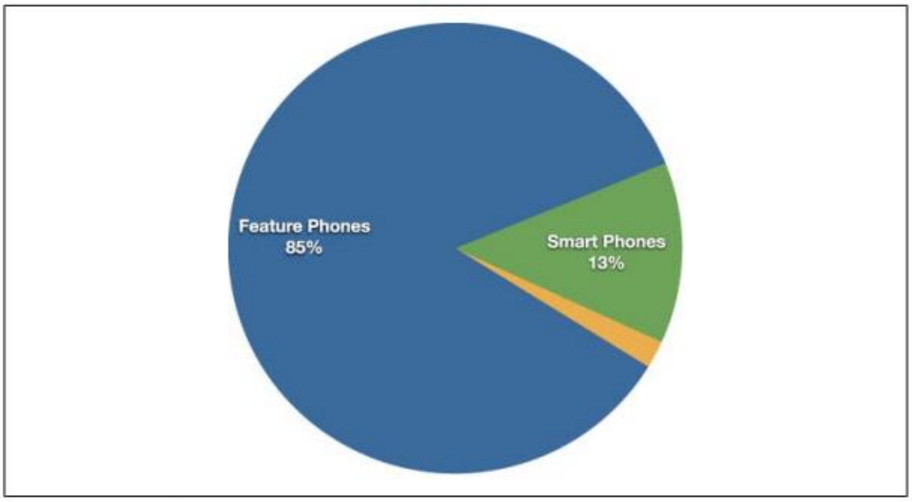


Figure 1-5. Breakdown of devices

Most mobile devices are subsidized in some form or another. Operators sell devices at a severely discounted price, often one-third or less of the actual cost of the device. This enables the operators to lock the devices to their networks. They can then preload onto the device content and services that are beneficial to themselves in exchange for lower price points, encouraging subscribers to upgrade to new devices with new capabilities. Subsidization means that devices need to be provisioned (or customized) to operators' individual requirements. Provisioning dramatically increases the number of devices released every year, with each device being slightly different from the other.

The sheer number of devices is both a blessing and a curse to the mobile industry. On the one hand, the magnitude of the mobile market is huge. It is one of the largest digital mediums mankind has ever seen. On the other hand, so many devices means adapting to those devices—not to mention painful and costly development cycles.

This brings us to the greatest challenge the mobile ecosystem currently faces: device fragmentation, a term used to describe how mobile devices interpret industry specifications differently, causing different mobile devices to display content inconsistently. Despite what we may know or have heard, we can take a deep breath and relax. Device fragmentation is a topic we will clear up completely in the following chapters.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

1.5.1 Platforms

A mobile platform's primary duty is to provide access to the devices. To run software and services on each of these devices, we need a platform, or a core programming language in which all of our software is written. Like all software platforms, these are split into three categories: licensed, proprietary, and open source.

Licensed

Licensed platforms are sold to device makers for nonexclusive distribution on devices. The goal is to create a common platform of development Application Programming Interfaces (APIs) that work similarly across multiple devices with the least possible effort required to adapt for device differences, although this is hardly reality.

Following are the licensed **platforms:**

Platforms	Description about the mobile Platforms
Java Micro Edition (Java ME)	Formerly known as J2ME, Java ME is by far the most predominant software platform of any kind in the mobile ecosystem. It is a licensed subset of the Java platform and provides a collection of Java APIs for the development of software for resource constrained devices such as phones.
Binary Runtime Environment for Wireless (BREW)	BREW is a licensed platform created by Qualcomm for mobile devices, mostly for the U.S. market. It is an interface-independent platform that runs a variety of application frameworks, such as C/C++, Java, and Flash Lite.
Windows Mobile	Windows Mobile is a licensable and compact version of the Windows operating system, combined with a suite of basic applications for mobile devices that is based on the Microsoft Win32 API.

LiMo	LiMo is a Linux-based mobile platform created by the LiMo Foundation. Although Linux is open source, LiMo is a licensed mobile platform used for mobile devices. LiMo includes SDKs for creating Java, native, or mobile web applications using the WebKit browser framework

Proprietary

Proprietary platforms are designed and developed by device makers for use on their devices. They are not available for use by competing device makers. These include:

Palm	Palm uses three different proprietary platforms. Their first and most recognizable is the Palm OS platform based on the C/C++ programming language; this was initially developed for their Palm Pilot line, but is now used in low-end smartphones such as the Centro line. As Palm moved into higher-end smartphones, they started using the Windows Mobile-based platform for devices like the Treo line. The most recent platform is called webOS, is based on the WebKit browser framework, and is used in the Prē line.
BlackBerry	Research in Motion maintains their own proprietary Java-based platform, used exclusively by their BlackBerry devices.
iPhone	Apple uses a proprietary version of Mac OS X as a platform for their iPhone and iPod touch line of devices, which is based on Unix.

Open Source

Open source platforms are mobile platforms that are freely available for users to download, alter, and edit. Open source mobile platforms are newer and slightly controversial, but they are increasingly gaining traction with device makers and developers. Android is one of these platforms. It is developed by the Open Handset Alliance, which is spearheaded by Google. The Alliance seeks to develop an open source mobile platform based on the Java programming language.

1.5.2 Operating Systems

It used to be that if a mobile device ran an operating system, it was most likely considered a smartphone. But as technology gets smaller, a broader set of devices supports operating systems.

Operating systems often have core services or toolkits that enable applications to talk to each other and share data or services. Mobile devices without operating systems typically run "walled" applications that do not talk to anything else.

Although not all phones have operating systems, the following are some of the most common:

Operating system	**Features of operating system**
Symbian	Symbian OS is a open source operating system designed for mobile devices, with associated libraries, user interface frameworks, and reference implementations of common tools.
Windows	Mobile Windows Mobile is the mobile operating system that runs on top of the Windows Mobile platform.
Palm OS	Palm OS is the operating system used in Palm's lower-end Centro line of mobile phones.
Linux	The open source Linux is being increasingly used as an operating system to power smartphones, including Motorola's RAZR2.
Mac OS	X A specialized version of Mac OS X is the operating system used in Apple's iPhone and iPod touch.
Android	Android runs its own open source operating system, which can be customized by users, operators and device manufacturers.

We might notice that many of these operating systems share the same names as the platforms on which they run. Mobile operating systems are often bundled with the platform they are designed to run on.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

1.6 Mobile Applications

A mobile application, also referred to as a mobile app or simply an app, is a computer program or software application designed to run on a mobile device such as a phone, tablet, or watch. Apps were originally intended for productivity assistance such as email, calendar, and contact databases, but the public demand for apps caused rapid expansion into other areas such as mobile games, factory automation, GPS and location-based services, order-tracking, and ticket purchases, so that there are now millions of apps available.

Apps are generally downloaded from application distribution platforms which are operated by the owner of the mobile operating system, such as the App Store (iOS) or Google Play Store. Some apps are free, and others have a price, with the profit being split between the application's creator and the distribution platform. Mobile applications often stand in contrast to desktop applications which are designed to run on desktop computers, and web applications which run in mobile web browsers rather than directly on the mobile device.

Native app

All apps targeted toward a particular mobile platform are known as native apps. Therefore, an app intended for Apple device do not run in Android devices. As a result, most businesses develop apps for multiple platforms. The main purpose for creating such apps is to ensure best performance for a specific mobile operating system.

While developing native apps, professionals incorporate best-in-class user interface modules. This accounts for better performance, consistency and good user experience. Users also benefit from wider access to application programming interfaces and make limitless use of all apps from the particular device. Further, they also switch over from one app to another effortlessly.

Hybrid app

The concept of the hybrid app is a mix of native and web-based apps. Apps developed using Xamarin, React Native, Sencha Touch and other similar technology fall into this category.

These are made to support web and native technologies across multiple platforms. Moreover, these apps are easier and faster to develop. It involves use of single code base which works in multiple mobile operating systems.

Despite such advantages, hybrid apps exhibit lower performance. Often, apps fail to bear the same look-and-feel in different mobile operating systems.

Web-based app

A web-based app is coded in HTML5, CSS or JavaScript. Internet access is required for proper behavior and user-experience of this group of apps.

These apps may capture minimum memory space in user devices compared to native and hybrid apps. Since all the personal databases are saved on the Internet servers, users can fetch their desired data from any device through the Internet.

1.6.1 Application Frameworks

Often, the first layer the developer can access is the application framework or API released by one of the companies mentioned already. The first layer that we have any control over is the choice of application framework.

Application frameworks often run on top of operating systems, sharing core services such as communications, messaging, graphics, location, security, authentication, and many others.

Application frameworks	Decription
Java	Applications written in the Java ME framework can often be deployed across the majority of Java-based devices, but given the diversity of device screen size and processor power, cross-device deployment can be a challenge. Most Java applications are purchased and distributed through the operator, but they can also be downloaded and installed via cable or over the air.
S60	The S60 platform, formerly known as Series 60, is the application platform for devices that run the Symbian

	OS. S60 is often associated with Nokia devices—Nokia owns the platform—but it also runs on several non-Nokia devices. S60 is an open source framework. S60 applications can be created in Java, the Symbian C++ framework, or even Flash Lite.
BREW	Applications written in the BREW application framework can be deployed across the majority of BREW-based devices, with slightly less cross-device adaption than other frameworks. However BREW applications must go through a costly and timely certification process and can be distributed only through an operator.
Flash Lite	Adobe Flash Lite is an application framework that uses the Flash Lite and ActionScript frameworks to create vector-based applications. Flash Lite applications can be run within the Flash Lite Player, which is available in a handful of devices around the world. Flash Lite is a promising and powerful platform, but there has been some difficulty getting it on devices. A distribution service for applications written in Flash Lite is long overdue.
Windows Mobile	Applications written using the Win32 API can be deployed across the majority of Windows Mobile-based devices. Like Java, Windows Mobile applications can be downloaded and installed over the air or loaded via a cable-connected computer.
Cocoa Touch	Cocoa Touch is the API used to create native applications for the iPhone and iPod touch. Cocoa Touch applications must be submitted and certified by Apple before being included in the App Store. Once in the App Store, applications can be purchased, downloaded, and installed over the air or via a cable-connected computer.
Android SDK	The Android SDK allows developers to create native applications for any device that runs the Android platform. By using the Android SDK, developers can write applications in C/C++ or use a Java virtual machine included in the OS that allows the creation of applications with Java, which is more common in the mobile ecosystem.
Web Runtimes (WRTs)	Nokia, Opera, and Yahoo! provide various Web Runtimes, or WRTs. These are meant to be miniframeworks, based on web standards, to create

	mobile widgets. Both Opera's and Nokia's WRTs meet the W3C-recommended specifications for mobile widgets. Although WRTs are very interesting and provide access to some device functions using mobile web principles, I've found them to be more complex than just creating a simple mobile web app, as they force the developer to code within an SDK rather than just code a simple web app. And based on the number of mobile web apps written for the iPhone versus the number written for other, more full-featured WRTs, It is a move in the right direction.
WebKit	With Palm's introduction of webOS, a mobile platform based on WebKit, and given its predominance as a mobile browser included in mobile platforms like the iPhone, Android, and S60, and that the vast majority of mobile web apps are written specifically for WebKit, I believe we can now refer to WebKit as a mobile framework in its own right. WebKit is a browser technology, so applications can be created simply by using web technologies such as HTML, CSS, and JavaScript. WebKit also supports a number of recommended standards not yet implemented in many desktop browsers. Applications can be run and tested in any WebKit browser, desktop, or mobile device
The Web	The Web is the only application framework that works across virtually all devices and all platforms. Although innovation and usage of the Web as an application framework in mobile has been lacking for many years, increased demand to offer products and services outside of operator control, together with a desire to support more devices in shorter development cycles, has made the Web one of the most rapidly growing mobile application platforms to date.

1.6.2 Applications

Application frameworks are used to create applications, such as a game, a web browser, a camera, or media player. Although the frameworks are well standardized, the devices are not. The largest challenge of deploying applications is knowing the specific device attributes and capabilities.

For example, if we are creating an application using the Java ME application framework, we need to know what version of Java ME the device supports, the screen dimensions, the processor power, the graphics capabilities, the number of buttons it has, and how the buttons are oriented. Multiply that by just a few additional handsets and we have hundreds of variables to consider when building an application. Multiply it by the most popular handsets in a single market and we can easily have a thousand variables, quickly dooming our application's design or development.

Although mobile applications can typically provide an excellent user experience, it almost always comes at a fantastic development cost, making it nearly impossible to create a scalable product that could potentially create a positive return on investment.

A common alternative these days is creating applications for only one platform, such as the iPhone or Android. By minimizing the number of platforms the developer has to support and utilizing modern application frameworks, the time and cost of creation go down significantly. This strategy may be perfectly acceptable to many, but what about the rest of the market? Surely people without a more costly smartphone should be able to benefit from mobile applications, too.

Many see the web browser as the solution to this problem and the savior from the insanity of deploying multidevice applications. The mobile web browser is an application that renders content that is device-, platform-, and operating-system-independent. The web browser knows its limitations, enabling content to scale gracefully across multiple screen sizes. However, like all applications, mobile web browsers suffer from many of the same device fragmentation problems.

We could consider the Motorola RAZR to be the epitome of the mobile ecosystem of yesterday. It's been provisioned to numerous operators around the world. It's the perfect example not just of how crazy deploying mobile applications to devices can be, but also of just how bad mobile web browser fragmentation can be. It is a highly prolific device and one that is often recommended for people to support, due to its market penetration. But that is much easier said than done.

If we look at the WURFL database (an open source device repository that is discussed later in this book), we can see that the V3, the real name of the RAZR, has an Openwave 6.2.3.2 web browser. The

V3/I/R had the Openwave 6.2.3.4.C.1.109 browser; the V3M/V9M had the Teleca Obigo 4.0 browser; the V3X had the Openwave 6.2.3.1.C. 1.112 browser; the V3M had the Openwave 6.2.3.1.C.1.115 browser; the V3XXI had the Opera 8.0 browser; and the V8 had the Opera 8.5 browser. This isn't even half the list!

From the consumer and business perspective, these are all Motorola RAZRs. But in terms of supporting the RAZR, these might as well be seven different devices. Each of these RAZRs carries very different versions of common applications, each customized for the operator on which they are intended to work. When a device is sold to an operator, it is provisioned (customized) to their requirements. This means the operators will often put customized applications on each of the devices sold. With the example of the RAZR, every operator had it and every operator put a different web browser on it. To make matters worse, the RAZRs, like most phones, are not field-refreshable, meaning that we can't update the software, upgrade the applications, or eliminate bugs.

For example, if a device manufacturer makes a device called the MDv1, they must strike a deal with an operator if they want to preload an operator store application, a different web browser, and bowling game. The device is sold as the MDv1.1. The operator sells the devices, or worse, gives them away for free. A couple hundred thousand of them go out into the marketplace before a glitch in the hardware is detected, such as dropped calls. Because the device cannot be upgraded by cable or over the air, the operator stops selling the MDv1.1, but seeing that they have a hit, they quickly replace it with the MDv1.1.1. The whole process is repeated as it is provisioned to each operator. Suddenly, there is an MDv1.2, an MDv1.3, an MDv1.4, and so on. Then we have the next generations—the MDv1.2.1, the MDv1.3.1, the MDv1.4.1, and so on, spreading like a virus. This is essentially what causes device fragmentation, making application development a costly and timely endeavor.

1.6.3 Services

Finally, we come to the last layer in the mobile ecosystem: services. Services include tasks such as accessing the Internet, sending a text message, or being able to get a location—basically, anything the user is trying to do. What makes the mobile environment such a complicated space to design and develop for are these layers, which the user must wade through in order to accomplish a simple task like "I want to send a text message," "I want to get on the Web," and "I want to access Google." The

user has so many opportunities for failure that creating a valuable experience is virtually impossible.

How do everyday people use their mobile devices? What are their impressions of the mobile web? Here is some of the consistent feedback we might hear if we ask these questions:

- "It's crap."
- "I don't use the mobile web, just because it's awful."
- "It costs too much."
- "I don't know where my browser is."
- "I don't know how to enter the URL."
- "I want to go to Wikipedia, but I don't know what to do."
- "How do I check my email?"

That is, of course, only if we are old—past our early 30s. The wenger we are, the more likely we are rely on mobile services for daily information. Earlier generations— those born since the birth of the Internet—have a unique talent for being able to figure out complicated informational spaces. They are more patient with technology and more apt to explore new methods of accomplishing tasks. And although one day the weth of today will inherit the digital world, for the time being, the mobile ecosystem is a complicated, fragmented, political nightmare.

If you were an entrepreneur looking to create mobile services, knowing what you know about mobile, you would run away, fast. you would probably open a restaurant, which would likely have a higher chance of success. But we've already seen the future of mobile development, in the form of the iPhone. The iPhone attempts to solve many of the problems facing the mobile ecosystem, from how people interact with their phones, to where we buy our phones, to what type of applications we will pay money for, to the level of technology standards we can support on constrained devices. What makes the iPhone special is how it attempts change on virtually all fronts, something no other device, or company for that matter, has been able to do previously.

Now Apple has done it, the gates are wide open for anyone. People in the industry aren't as jaded anymore, and there is a feeling of excitement and optimism. Although many of the problems in the mobile ecosystem are yet to be fixed and we still have plenty of nonsense to contend with, we can see the light. We can see the path to innovation, to creating applications and services that can quite literally reach the entire planet and quite possibly change the world. It begins here right now.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

1.7 Check your progress Questions

1. The mobile ecosystem consisting of
   a) Network　　b) Operating systems
   c) Devices　　d) All the above

2. Which one of the following is the second layer of mobile ecosystem architecture?
   a) Network　　b) Operating systems
   c) Devices　　d) Services

3. Which one of the following is not a 3G network?
   a) W-CDMA　　b) UMTS
   c) UMTS-TDD　　d) GSM

4. Find out the odd one
   a) Symbian　　b) Windows
   c) Android　　d) JAVA

5. Mobile platforms are primary split into three categories. They are
   a) licensed, proprietary, and open source.
   b) licensed, customized, and open source.
   c) customized, exportable and open source.
   a) customized, prioritized and open source.

6. Application frameworks often run on top of ________.
   a) Network　　b) Operating systems
   c) Devices　　d) Services

1.8 Answer to Check your progress Questions

1. d) All the above
2. a) Network
3. d) GSM
4. d) JAVA
5. a) licensed, proprietary, and open source.
6. b) Operating systems

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

1.9 Summary

Mobile Ecosystem is collection of multiple operators, networks, devices, operating system, applications, process and services by companies. Operators can be referred to as mobile network operators, mobile service providers, wireless carriers, or simply mobile phone operators or cellular companies.

GSM, GPRS, EDGE, and HSCSD are 2G networks. W-CDMA, UMTS, UMTS-TDD, TD-CDMA, HSPA, HSDPA, and HSUPA are 3G networks.

Feature phones and Smartphones are mostly used Mobile devices around the world. A mobile platform's primary duty is to provide access to the devices. To run software and services on each of these devices, we need a platform, or a core programming language in which all of our software is written.

Java Micro Edition (Java ME), Binary Runtime Environment for Wireless (BREW), Windows Mobile, LiMo, are popular mobile platforms. Proprietary platforms are designed and developed by device makers for use on their devices. Palm, BlackBerry, and iPhone are Proprietary platforms. Open source platforms are mobile platforms that are freely available for users to download, alter, and edit. Android is one of these platforms.

Operating systems often have core services or toolkits that enable applications to talk to each other and share data or services. Symbian, Windows, Palm OS, Linux, Mac OS, Android are popular operating systems.

Application frameworks often run on top of operating systems, sharing core services such as communications, messaging, graphics, location, security, authentication, and many others. Application frameworks are used to create applications, such as a game, a web browser, a camera, or media player.

Java, S60, BREW, Flash Lite, Windows Mobile, Cocoa Touch, Android SDK, Web Runtimes, WebKit, and The Web are The Application frameworks. Services include tasks such as accessing the Internet, sending a text message, or being able to get a location.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

1.10 Key words

Mobile Ecosystem, Operators, Networks, Devices, Platforms, Operating systems, Application frameworks ,Applications, Services, Symbian, Windows, Palm OS, Linux, Mac OS, Android.

1.11 Self Assessment Questions

Short Answer Questions

1. What do we mean by mobile ecosystems?
2. What are the types of Mobile Devices?
3. List the uses of mobile application.
4. How the networks provide support to devices?
5. List out some platforms to build mobile application.
6. Write the various mobile operating systems.
7. Give the reasons to build Mobile Application.
8. Write few services of mobile ecosystems.

Long Answer Questions

1. Explain The Mobile Ecosystem with the following concepts
    a) Operators
    b) Networks
    c) Devices
    d) Platforms
    e) Operating systems
    f) Application frameworks
    g) Applications
    h) Services
2. Discuss the application frameworks of the mobile ecosystem in detail.
3. Explain the device details covered in mobile ecosystem.
4. Draw the mobile ecosystem architecture and explain each component clearly.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

1.12 Further readings

1. Mobile Design and Development by Brian Fling, O'Reilly Media, Inc 2009

2. J2ME: The Complete Reference, James Keogh, Tata McGrawHill 2003

3. H. Lashkari, M. Moradhaseli, Mobile operating systems and programming: mobile communications, VDM Verlag Dr. Muller, 2011

4. http://opengardensblog.futuretext.com/archives/2008/06/iphone_vs_symbi_1.html

5. http://www.isecpartners.com/files/iSEC_Securing_Android_Apps.pdf

6. http://www.ibm.com/developerworks/architecture/library/wi-arch23.html

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

UNIT 2 MOBILE DEVICES PROFILES

2. 1 Contents of the unit

2.2 Introduction

A mobile device plays a vital role in every human life. There are several mobile application medium types. More than billions of mobile applications such as gaming app, utility app, and location based apps are used by people.

The most basic mobile application is an Short message service(SMS) application. It can be integrated with many other mobile application types. There are lots of SMS mobile alert applications.

A mobile game is a game played on a feature phone, smart phone, tablet, smart watch, PDA, portable media player or graphing calculator. The earliest known game on a mobile phone was a Tetris.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

The mobile web refers to browser-based World Wide Web services accessed from handheld mobile devices, such as smart phones or feature phones, through a mobile or other wireless network.

A location-based service is the name for a general class of policies in software-level services that provide for accessing data, files, pipes, memory objects, streams and other or online services.

Informative apps provide information for best learning. There are several free learning best apps on Android and iOS Platform. Enterprise apps target a wide range of industries and can integrate with office, purchase, sales and customer service processes.

2.3. Objectives

- To study varies categories of mobile applications.
- Be familiar with types of mobile medium.
- To understand the basic concepts of SMS and mobile apps using SMS service.
- To acquire more knowledge in mobile websites.
- Be familiar with native mobile apps.
- To gain knowledge about different location based services.
- To study different Informative and Enterprise apps

2.4 Mobile Devices Profiles

Short for Mobile Information Device Profile (MIDP). MIDP is a set of J2ME APIs that define how software applications interface with cellular phones. Mobile Information Device Profile (MIDP) is a specification published for the use of Java on embedded devices such as mobile phones and PDAs. MIDP is part of the Java Platform, Micro Edition (Java ME) framework and sits on top of Connected Limited Device Configuration (CLDC), a set of lower level programming interfaces. MIDP was developed under the Java Community Process.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

2.4.1 Categories of Mobile Applications

Introduction

The mobile ecosystem is a large and deep pool. Mobile applications aren't that much different from boats in this seafaring example. You have a number of choices in what medium you use to address your goals, each with their own pros and cons. Some are quick to create but accessible to fewer users. Others address a larger market, but are far more complex and costly.

Mobile Application Medium Types

The mobile medium type is the type of application framework or mobile technology that presents content or information to the user. It is a technical approach regarding which type of medium to use; this decision is determined by the impact it will have on the user experience. The technical capabilities and capacity of the publisher also factor into which approach to take.

We discussed the common mobile platforms in terms of how they factor in the larger mobile ecosystem. Now we will look deeper into each of these platforms from a more tactical perspective, unpacking them, so to speak, to see what is inside.

Categories of Mobile Application medium types are listed below:

- SMS
- Mobile Websites
- Mobile Web Widgets
- Mobile Web Applications
- Native Applications
- Games
- Mobile Application Media Matrix
- Application Context
- Utility Context
- Locale Context
- Informative Applications
- Productivity Application Context
- Immersive Full-Screen Applications
- Application Context Matrix

There are three types of popular apps as shown in figure 2-1

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

1. Native apps is easy to use and mostly used. Native mobile applications are completely coded in a specific programming language.

- iOS on Objective-C or Swift
- Android on Java
- Windows Phone on Net

2. Hybrid apps for all platforms altogether with Xamarin, React Native, Ionic, Angular Mobile Sencha Touch etc.

3. Web apps as responsive versions of website to work on any mobile device.

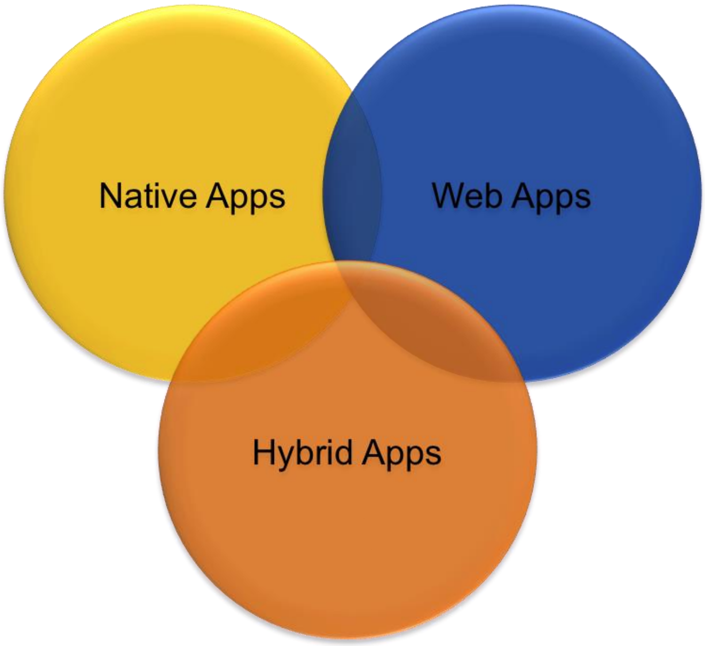


Figure 2-1 Types of Mobile Application

2.4.2 SMS

SMS (short message service) is a text messaging service component of most telephone, Internet, and mobile device systems. It uses standardized communication protocols to enable mobile devices to exchange short text messages. An intermediary service can facilitate a text-to-voice conversion to be sent to landlines.

The Short Message Service (SMS) allows the exchange of short messages between a mobile station and the wireless system, and between the wireless system and an external device capable of transmitting and optionally receiving short messages.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

The external device may be a voice telephone, a data terminal or a short message entry system. The Short Message Service consists of message entry features, administration features, and message transmission capabilities. These features are distributed between a wireless system and the SMS message center (MC) that together make up the SMS system.

The message center may be either separate from or physically integrated into the wireless system. Short message entry features are provided through interfaces to the message center and the mobile station. Senders use these interfaces to enter short messages, intended destination addresses, and various delivery options.

The most basic mobile application you can create is an SMS application. Although it might seem odd to consider text messages applications, they are nonetheless a designed experience. Given the ubiquity of devices that support SMS, these applications can be useful tools when integrated with other mobile application types.

Typically, the user sends a single keyword to a five-digit short code in order to return information or a link to premium content. For example, sending the keyword “freebie” to a hypothetical short code “12345” might return a text message with a coupon code that could be redeemed at a retail location, or it could include a link to a free ringtone.

SMS applications can be both “free,” meaning that there is no additional charge beyond the text message fees an operator charges, or “premium,” meaning that you are charged an additional fee in exchange for access to premium content.

The most common uses of SMS applications are mobile content, such ringtones and images, and to interact with actual goods and services. Some vending machines can dispense beverages when you send them an SMS; SMS messages can also be used to purchase time at a parking meter or pay lot.

A great example of how SMS adds incredible value would be Twitter, where users can receive SMS alerts from their friends and post to their timeline from any mobile device, or the SMS-to-Billboard that BBC World News put up in Midtown Manhattan (Figure 2-2).



Figure 2-2. An SMS application to interact with a billboard in Manhattan

SMS messaging can be used as a marketing tool. An example is an SMS newsletter system. After signing up, the user will receive SMS text messages about the latest discounts and products of the company. If the user has any questions or comments, he/she can send a text message back with the questions or comments in it. The company may include its phone number in the SMS newsletter so that the user can talk to the customer service staff directly if he/she wants to do so.

In a remote system monitoring application, a program (sometimes with the help of a group of sensors) is constantly monitoring the status of a remote system. If a certain condition is satisfied, the program will send a text message to the system administrator to notify him/her of the situation. For example, a program may be written to "ping" a server regularly. If no response is received from the server, the program can send an SMS alert to the system administrator to notify him/her that the server may be hanged.

A chat application is another kind of person-to-person text messaging application that allows a group of people to exchange SMS text messages interactively. In a chat application, all SMS text messages sent and received are displayed on the mobile phone's screen in order of date and time.

SMS banking is a form of mobile banking. It is a facility used by some banks or other financial institutions to send messages (also called notifications or alerts) to customers' mobile phones using SMS messaging,

or a service provided by them which enables customers to perform some financial transactions using SMS.

SMS banking services may use either push and pull messages. Push messages are those that a bank sends out to a customer's mobile phone, without the customer initiating a request for the information.

Typically, a push message could be a mobile marketing message or an alert of an event which happens in the customer's bank account, such as a large withdrawal of funds from an ATM or a large payment involving the customer's credit card, etc. It may also be an alert that some payment is due, or that an e-statement is ready to be downloaded.

Pros

The pros of SMS applications include:

- They work on any mobile device nearly instantaneously.
- They're useful for sending timely alerts to the user.
- They can be incorporated into any web or mobile application.
- They can be simple to set up and manage.

Cons

The cons of SMS applications include:

- They're limited to 160 characters.
- They provide a limited text-based experience.
- They can be very expensive.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

2.4.3 Mobile Websites

As you might expect, a mobile website is a website designed specifically for mobile devices, not to be confused with viewing a site made for desktop browsers on a mobile browser. Mobile websites are characterized by their simple "drill-down" architecture, or the simple presentation of navigation links that take you to a page a level deeper, as shown in Figure 2-3.



Figure 2-3. An example of a mobile website

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Mobile websites often have a simple design and are typically informational in nature, offering few—if any—of the interactive elements you might expect from a desktop site. Mobile websites have made up the majority of what we consider the mobile web for the past decade, starting with the early WML-based sites (not much more than a list of links) and moving to today's websites, with a richer experience that more closely resembles the visual aesthetic users have come to expect with web content.

Though mobile websites are fairly easy to create, they fail to display consistently across multiple mobile browsers—a trait common to all mobile web mediums.

The mobile web has been gradually increasing in usage over the years in most major markets, but the limited experience offered little incentive to the user. Many compare the mobile web to a 10-year-old version of the Web: slow, expensive to use, and not much to look at.

As better mobile browsers started being introduced to device platforms like the iPhone and Android, the quality of mobile websites began to improve dramatically, and with it, usage improved. For example, in just one year, the U.S. market went from being just barely in the top five consumers of the mobile web to number one, largely due to the impact of the iPhone alone.

Pros

The pros of mobile websites are:

- They are easy to create, maintain, and publish.
- They can use all the same tools and techniques you might already use for desktop sites.
- Nearly all mobile devices can view mobile websites.

Cons

The cons of mobile websites are:

- They can be difficult to support across multiple devices.
- They offer users a limited experience.
- Most mobile websites are simply desktop content reformatted for mobile devices.
- They can load pages slowly, due to network latency.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

2.4.4 Mobile Web Widgets

Largely in response to the poor experience provided by the mobile web over the years, there has been a growing movement to establish mobile widget frameworks and platforms. For years the mobile web user experience was severely underutilized and failed to gain traction in the market, so several operators, device makers, and publishers began creating widget platforms (Figure 2-4) to counter the mobile web's weaknesses.

Trying to define what exactly a mobile web widget is and how it is different from the other mobile web media is a question for the ages. We initially saw mobile web widgets as another attempt by the mobile industry to hype a technology that no one wants.

We liked to quiz mobile web widget advocates about what makes mobile web widgets different than what we can do with the mobile web. We will never get a straight answer. So in order to define a mobile web widget, I followed some advice from my dad: "When in doubt, look it up in the dictionary." Here was the answer in Webster's Dictionary:

A component of a user interface that operates in a particular way.

- The ever-trusty Wikipedia defines a web widget this way:

  A portable chunk of code that can be installed and executed within any separate HTMLbased web page by an end user without requiring additional compilation.

- Between these two definitions is a better answer:

  A mobile web widget is a standalone chunk of HTML-based code that is executed by the end user in a particular way.



Figure 2-4. An example mobile web widget

Basically, mobile web widgets are small web applications that can't run by themselves; they need to be executed on top of something else. I think one reason for all the confusion around what is a mobile web widget is that this definition can also encompass any web application that runs in a browser. Opera Widgets, Nokia Web RunTime (WRT), Yahoo! Blueprint, and Adobe Flash Lite are all examples of widget platforms that work on a number of mobile handsets.

Using a basic knowledge of HTML (or vector graphics in the case of Flash), you can create compelling user experiences that tap into device features and, in many cases, can run while the device is offline. Widgets, however, are not to be confused with the utility application context, a user experience designed around short, task-based operations.

Pros

The pros of mobile web widgets are:

- They are easy to create, using basic HTML, CSS, and JavaScript knowledge.
- They can be simple to deploy across multiple handsets.
- They offer an improved user experience and a richer design, tapping into device features and offline use.

Cons

The cons of mobile web widgets are:

- They typically require a compatible widget platform to be installed on the device.
- They cannot run in any mobile web browser.
- They require learning additional proprietary, non-web-standard techniques.

Mobile Web Applications

Mobile web applications are mobile applications that do not need to be installed or compiled on the target device. Using XHTML, CSS, and JavaScript, they are able to provide an application-like experience to the end user while running in any mobile web browser. By "application-like" experience, I mean that they do not use the drill-down or page metaphors in which a click equals a refresh of the content in view. Web applications allow users to interact with content in real time, where a click or touch performs an action within the current view.

The history of how mobile web applications came to be so commonplace is interesting, and is one that I think can give us an understanding of how future mobile trends can be assessed and understood. Shortly after the explosion of Web 2.0, web applications like Facebook, Flickr, and Google Reader hit desktop browsers, and there was discussion of how to bring those same web applications to mobile devices. The Web 2.0 movement brought user-centered design principles to the desktop web, and those same principles were sorely needed in the mobile web space as well.

The challenge, as always, was device fragmentation. The mobile browsers were years behind the desktop browsers, making it nearly impossible for a mobile device to render a comparable experience. While XHTML support had become fairly commonplace across devices, the

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.
rendering of CSS2 was wildly inconsistent, and support for JavaScript, necessary or simple DHTML, and Ajax was completely nonexistent.

With the introduction of the first iPhone, we saw a cataclysmic change across the board. Using WebKit, the iPhone could render web applications not optimized for mobile devices as perfectly usable, including DHTML- and Ajax-powered content. Developers quickly got on board, creating mobile web applications optimized mostly for the iPhone (Figure 2-5). The combination of a high-profile device with an incredibly powerful mobile web browser and a quickly increasing catalog of nicely optimized experiences created the perfect storm the community had been waiting for.



Figure 2-5. The Facebook mobile web app

Usage of the mobile web exploded with not just users of the iPhone, but users of other handsets, too. Because web applications being created for the iPhone were based on web standards, they actually worked

reasonably well on other devices. Operators and device makers saw that consumers wanted not just the mobile web on their handsets, but the regular Web, too.

In less than a year, we saw a strong unilateral move by all operators and devices makers to put better mobile web browsers in their phones that could leverage this new application medium. We have not seen such rapid innovation in mobile devices since the inclusion of cameras.

The downside, of course, like all things mobile-web-related, is that not all devices support the capability to render mobile web applications consistently. However, we do see a prevalent trend that the majority of usage of the mobile web is coming from the devices with better browsers, in some markets by a factor of 7:1. So although creating a mobile web application might not reach all devices, it will reach the devices that create the majority of traffic.

Pros

The pros of mobile web applications are:

• They are easy to create, using basic HTML, CSS, and JavaScript knowledge.

- They are simple to deploy across multiple handsets.
- They offer a better user experience and a rich design, tapping into device features and offline use.
- Content is accessible on any mobile web browser.

Cons

The cons of mobile web applications are:

- The optimal experience might not be available on all handsets.
- They can be challenging (but not impossible) to support across multiple devices.
- They don’t always support native application features, like offline mode, location lookup, filesystem access, camera, and so on.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

2.5 Native Applications

The next mobile application medium is the oldest and the most common; it is referred to as native applications, which is actually a misnomer because a mobile web app or mobile web widget can target the native features of the device as well. These applications actually should be called "platform applications," as they have to be developed and compiled for each mobile platform.

These native or platform applications are built specifically for devices that run the platform in question. The most common of all platforms is Java ME (formerly J2ME). In theory, a device written as a Java ME MIDlet should work on the vast majority of feature phones sold around the world. The reality is that even an application written as a Java ME MIDlet still requires some adaptation and testing for each device it is deployed on.

In the smartphone space, the platform SDKs get much more specific. Although many smartphones are also powered by Java, an operating system layer and APIs added to allow developers to more easily offload complex tasks to the API instead of writing methods from scratch. In addition to Java, other smartphone programming languages include versions of C, C++, and Objective-C (Figure 2-6).

Creating a platform application means deciding which devices to target, having a means of testing and certification, and a method to distribute the application to users. The vast majority of platform applications are certified, sold, and distributed either through an operator portal or an app store. It is possible to create a Java ME MIDlet application and publish it for free on the Web, but it is rarely done.

Because platform applications sit on top of the platform layer, they can tap into the majority of the device features, working online or offline, accessing the location and the filesystem—and if there's camera on the device, then you can probably do something with it as well. Hence the need for certification before the application is distributed, to ensure that no one distributes an application that steals a user's personal data or maliciously uses the device to spread viruses.



Figure 2-6. A native application in the iPhone

However, if you exclude games, the majority (by some estimates, as much as 70 percent) of native applications in use today could be created with a little bit of XHTML, CSS, and JavaScript—in other words, a mobile web application, with little or no feature loss to the user. The advantage is that a mobile application can be developed faster, will work on more devices, require less testing, and be updated more transparently than a native application, which requires third-party certification and publishing in order to get on users' devices. All of these aspects are highly desired in the platform application space. The downside is that it requires a fast and capable mobile web browser that supports offline data and access to device features like location.

Pros

The pros of native applications include:

- They offer a best-in-class user experience, offering a rich design and tapping into device features and offline use.
- They are relatively simple to develop for a single platform.
- You can charge for applications.

Cons

The cons of native applications include:

- They cannot be easily ported to other mobile platforms.
- Developing, testing, and supporting multiple device platforms is incredibly costly.
- They require certification and distribution from a third party that you have no control over.
- They require you to share revenue with the one or more third parties.

2.5.1 Games

A mobile game is a game played on a feature phone, smart phone, tablet, smart watch, PDA, portable media player or graphing calculator. The earliest known game on a mobile phone was a Tetris.

The interesting and entertained mobile medium is games, the most popular of all media available to mobile devices. Technically games are really just native applications that use the similar platform SDKs to create immersive experiences (Figure 2-7). But I treat them differently from native applications for two reasons: they cannot be easily duplicated with web technologies, and porting them to multiple mobile platforms is a bit easier than typical platform-based applications.

Although you can do many things with a powerful mobile web browser, creating an immersive gaming experience is not one of them—at least not yet. Seeing as how we have yet to see these types of gaming experiences appear on the desktop using standard web technologies,

We are still a few years out from seeing them on mobile devices. Adobe's Flash and the SVG (scalable vector graphics) standard are the only way to do it on the Web now, and will likely be how it is done on mobile devices in the future, the primary obstacle being the performance of the device in dealing with vector graphics.

The reason games are relatively easy to port ("relatively" being the key word), is that the bulk of the gaming experience is in the graphics and actually uses very little of the device APIs. The game mechanics are the only thing that needs to adapted to the various platforms. Like in console gaming, there are a great number of mobile game porting shops that can quickly take a game written in one language and port it to another.

These differences, in my mind, are what make mobile games stand apart from all other application genres—their capability to be unique and difficult to duplicate in another application type, though the game itself is relatively easy to port. Looking at this model for other application areas—namely, the mobile web—could provide helpful insight into how we create the future of mobile web applications.



Figure 2-7. An example game for the iPhone

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Pros

The pros of game applications are:

- They provide a simple and easy way to create an immersive experience.
- They can be ported to multiple devices relatively easily.

Cons

The cons of game applications are:

- They can be costly to develop as an original game title.
- They cannot easily be ported to the mobile web.

Mobile Application Media Matrix

To aid in comparing and contrasting which of these mobile application media is best for your mobile product, we placed them into a matrix (Table 2.1).

Matrix	Device support	Complexity	User experience	Language	Offline support	Device features
SMS	All	Simple	Limited	N/A	No	None
Mobile websites	All	Simple	Limited	HTML	No	None
Mobile web widgets	Some	Medium	Great	HTML	Limited	Limited
Mobile web applications	Some	Medium	Great	HTML, CSS, JavaScript	Limited	Limited
Native applications	All	Complex	Excellent	Various	Yes	Yes
Games	All	Complex	Excellent	Various	Yes	Yes

Application Context

Once your application medium is decided upon, it is time to look at the application context, or the appropriate type of application to present to the user in order for the user to process and understand the information presented and complete her goals. Where the application medium refers mostly to the technical approach of creating an application, the application context deals with the user experience.

Applications can be presented in a variety of ways, ranging from a simple task-based utility to an experience meant to consume the user's focus and attention. There of course is no right or wrong direction—only

what is best for your user. In fact, nothing says that you can't use multiple application contexts within the same application.

We just wouldn't recommend it unless you have really thought out the flow of your application, because typically it is best to present only one application context so as to avoid confusing the user. If you think it best for your app to mix contexts, then give the user the option to switch between them; for example, some smartphones allow for an orientation change, so if the device is rotated to landscape mode, your app switches from an informative view to a utility view, or maybe from a locale view to an immersive view.

2.5.2 Utility Apps

The most basic application context is the utility, or a simple user experience metaphor that is meant to address short, task-based scenarios. Information is meant to be presented in a minimal fashion, often using the least amount of user input as possible. An example of a utility might be a calculator, weather forecast, unit conversion, stocks, world clock, and so on. In each of these cases, the user enters a small amount of information into the utility, like a simple equation, a city, or a stock symbol, and either by performing a small background task or fetching information online, the utility is able to present data to the user in the desired context (Figure 2-8).



Figure 2-8. An example utility application

The goal of the utility is to give users at-a-glance information, therefore offering users a minimal design aesthetic, focusing the design around the content in view, and often using larger type and a sparse layout.

It would be easy to mistake utilities for widgets, given that widgets are a "component of a user interface that operates in a particular way." But utilities can be much more than widgets; they are not merely an extension of the user experience, but are applications in their own right that can establish their own look and feel separate from the established user experience.

Use utilities for short, simple tasks, at-a-glance information, when there is limited content to display, and when combined with an immersive context to create dual-mode applications.

Calculator provides simple and advanced mathematical functions in a beautifully designed app.

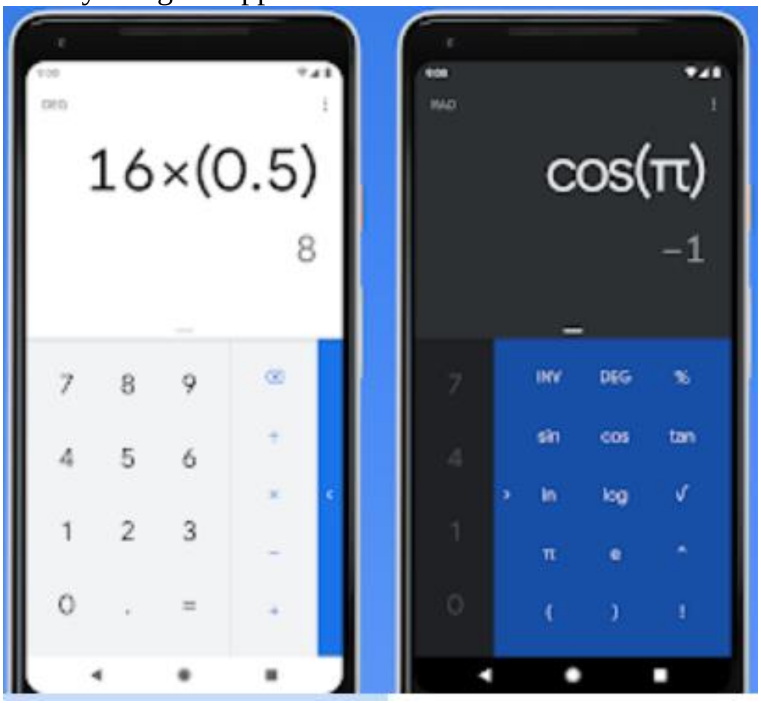


Figure 2-9. An example utility application – Calculator.

It performs basic calculations such as addition, subtraction, multiplication, and division. It also do scientific operations such as trigonometric, logarithmic, and exponential functions.

2.5.3 Location Based Services(LBS)

Locale Context The locale context is a newer application type that is still being defined by the mobile community, but we are certainly seeing some clear patterns of how to create locale applications (Figure 2-10). As more location information is being published online, and more devices add GPS to pinpoint the user's location, locale is becoming an excellent data point to pivot information around. For example, we can use location to display the cafés nearest to my current location.



Figure 2-10. An example locale application

Locale applications almost always have at least one thing in common: a map on which they plot the requested data points visually. At the very least, they list items in order of distance, with the nearest item first and the farthest last.

The user’s goal is to find information relative to his present location, and content should always be designed with this in mind. When creating locale apps, it is important to ensure that the user’s present location is always clearly identified, as well as a means of adding data to it. This could be another location, in the case of finding point-to-point

directions, or it could be a keyword query to find people, places, or things nearby.

Use locale applications for location-based applications, applications that might contain a dynamic map, and listing multiple location-based points of interest.

Location-Aware Mobile Computing

Mobility support of mobile wireless systems naturally leads to location-aware computing that encompasses a number of techniques, protocols, and enabling mobile wireless technologies and mobile devices utilizing an object's location information to provide augmented consumer- or business-oriented applications and services - namely, location-based services (LBSs). To mobile service providers, LBSs seem to be good candidates for value-added services that have the potential to grow significantly as more location-aware mobile devices and network capabilities are being used.

An object's location is indeed a key component of context, meaning that location-aware computing is actually concerned with making a mobile wireless system context aware in terms of location. Location-based services and applications are therefore the first step toward context awareness in a pervasive mobile computing environment. The role of smart phones in the context of location-aware computing is crucial, as a smart phone is very likely to have multiple wireless interfaces; thus, with some supporting software, a smart phone could be used as a location-sensing device for object localization.

The first issue is concerned with choosing the best location data model for a location-aware computing system, whereas localization is generally defined as involving location determination schemes and algorithms. Enabling technologies for localization in the wireless world are referred to as location sensing technologies, including global positioning systems (GPSs), cellular base stations, wireless local area networks (LANs), ultrasound, radio frequency identification (RFID) tags, and wireless sensor networks. An assortment of location sensing systems using one or more of these technologies has been developed. In particular, indoor location sensing is in great demand by many industry sectors, such as healthcare facilities and warehouses, yet numerous challenging issues remain to be addressed.

Localization in wireless sensor networks is another difficult but interesting issue, and location awareness is becoming a building block for

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.
many mobile applications and services. To foster future location-aware computing systems and location-based services, we explore location-related schemes and techniques used in emerging location-based services and applications.

Location Representation

In location-aware computing, the terms position and location are used frequently. In most cases they are interchangeable, but they do not always refer to the same thing. Position is defined as the geospatial place of an object, often represented by coordinates. Location has a more general, semantic meaning with respect to the position of an object; for example, the location of a person could be "room 113 of the engineering building." The process of determining an object's location is called localization, or positioning. Based on location information collected at different granularities over time and on the context of movement, object movement tracking and path projection can also be performed.

A core element of location-based services and location sensing systems is location representation. Localization algorithms generate data in the form of a location representation model, and applications and services must make location queries conforming to the selected location representation model. Depending on the circumstance of the application, different models can be used, such as the geometric model, symbolic model, and location graph model. A geometric model uses location properties of an object to pinpoint it in two-dimensional or three-dimensional space. The properties can be distance, angle, time, etc.

The coordinates in a geometric model can be either absolute coordinates, such as latitude, longitude, and altitude used by GPS, or relative coordinates, such as an application-specific grid coordinates. Geometric models are fairly simple to process mathematically in Euclidian space, but in some cases they may be overkill and may not provide truly useful information to an application, as raw coordinates are in effect meaningless without referencing objects such as rooms, buildings, and streets. Symbolic or semantic models seem to be a better solution to relative localization, which simply identifies an object as being within or close to some known objects.

For example, in an indoor location sensing system, a symbolic location representation could be "John Doe is now in room 3353 on the third floor of the engineering building." This type of location information

is more meaningful than the person's geospatial coordinates. The symbolic model does not require fine-grained position information of the object.

A third type of model, a location map, is based on a map of symbolic representations rather than geometric ones. It is essentially a conceptual combination of the other two models. Consequently, it is more complicated than the other two models because of the integration and mapping between the geometric location information and symbolic location information. Choosing a location representation model is largely dependent on the requirements of the location sensing system in terms of overall accuracy, location update frequency, and infrastructure cost.

Localization Techniques

Location sensing systems invariably employ some localization techniques or algorithms that generally require some sort of wireless signal measurement to compute an object's position. Localization techniques can be divided into three categories: triangulation, scene analysis, and proximity.

1. Triangulation: Triangulation utilizes geometric properties of a triangle to compute an object's position based on the positions of two or three vertices of a triangle. The two types of triangulation are lateration and angulation. Lateration relies on distance measurements from the object being localized to three reference points and distances between reference points, whereas angulation achieves the same task using angular or bearing measurements and a known distance between two reference points. Figure 2-11 shows these two basic techniques on a two-dimensional plane. In Figure 2.11a, three noncollinear reference points are required to obtain a total number of six distance measurements for lateration. For three-dimensional localization, four noncoplanar reference points are needed. In Figure 2-11b, only two reference points, one distance measurement between the two reference points, and two angular measurements are necessary for angulation. In three dimensions, an additional azimuth measurement (horizontal angle between a reference point and the unknown object) is necessary. Note that in angulation the magnetic north is often chosen as the base of angle measurement. Distance measurements for lateration in wireless communication are often conducted by measuring the time of flight (TOF), time of arrival (ToA), or angle of arrival (AOA) of wireless signals. Given the speed of a radio signal or a sound wave traveling in

space, the distance between a transmitter and a receiver can be computed by multiplying the speed and the time it takes to reach the receiver (TOF). Because the speed of light and radiowaves is extremely high (approximately 300,000 km/sec), TOF measurement must have a very high resolution to achieve reasonable accuracy. For example, in order to achieve a location accuracy of 5 to l Om, the clock resolution of a distance measurement must be at the level of 15 to 30 nsec. If a radio signal is unidirectional, the transmitter and the receiver must have their clocks synchronized to measure TOF. This is often difficult to realize in a distributed network environment. A solution to this problem is to enable precise clock synchronization among reference points but not the unknown object. Thus, the clock drift of the unknown object is an additional variable in lateration or angulation computations and requires an additional reference point.

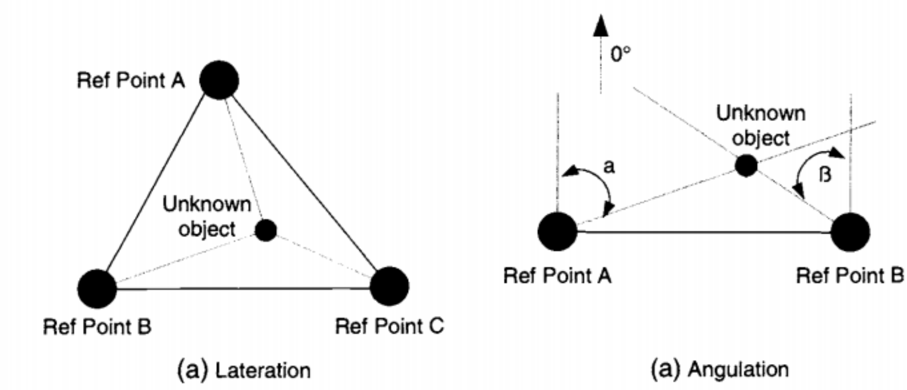


Figure 2-11 Triangulation techniques.

2. Scene analysis: Scene analysis for location sensing circumvents geometric computation but requires a map of the scene where known objects are located. The map is a dataset of features, such as signal strength across a space, and visual scenery taken from a vantage point. Static scene analysis, sometimes referred to as calibration, compares the observed features at a location to an existing feature map to identify the position in question. In differential scene analysis, changes between subsequent scenes are used to track the observer's position. In both cases, some imaging processing and information retrieval (IR) techniques are likely to be used. In addition, because of the inherent uncertainty in scene analysis, sophisticated statistical models such as Bayes filters have been introduced mostly in indoor location sensing system that use electronic features for scene analysis. Aside from asset or person localization, scene

analysis is used in wireless networks, mostly 802.11 wireless LANs, for site surveys to optimize access point channels and locations in a given location, utilizing onsite measurement along with floor plans. The technique can be also used in wireless intrusion detection systems (IDSs) to detect and estimate the location of rogue access points and possible wireless attacks. Generally, wireless IDSs are able to monitor and analyze wireless traffic in the network and identify abnormal traffic patterns from mobile stations and access points. Localization techniques such as proximity, triangulation, and scene analysis are common components of wireless IDS to help network administrators determine the location of rogue access points and mobile stations. Some commercial Wi-Fi location systems such as Ekahau and Airspace RF Fingerprinting have been developed using this approach scene analysis techniques. Ekahau (http://www.ekahau.com/) does not employ any propagation or triangulation methods that suffer from radiowave multipathing, scattering, and attenuation effects. Instead, a scene analysis method called site calibration is used for collecting radio network sample points from different site locations. Each sample point contains a statistical summary of the received signal intensity (RSSI) and related map coordinates. The limitation of this approach is that the map has to be updated whenever the underlying scene changes with respect to the selected featured being presented. In addition, building a map of features by sampling at many places is not an easy task. Hence, a balance has to be achieved between map granularity and map creation overhead. Researchers have been using robot to conduct this tedious work.

3. Proximity: Proximity-based location sensing techniques do not offer direct quantitative position of an unknown object. Instead, they only provide proximity information such as serving base stations of the unknown object. The main idea of these techniques is essentially binary distance measurement to known reference points in proximity. For example, by comparing signal strength at a set of wireless LAN access points from a wireless LAN device, a proximity base location sensing system is able to tell which access points are closer to the devices than others. Because those wireless access points were previously mapped to physical locations, the system can localize and track the device with respect to the signal scope of base stations encountered. Clearly the accuracy of such systems relies on the granularity of the base stations in the proximity determination. If a location-based application employs a symbolic location representation model, then proximity-based localization techniques

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

can be used to localize an unknown object with respect to symbolic locations of reference points in proximity.

Global Positioning System(GPS)

The GPS system consists of a total number of 24 GPS satellites orbiting the Earth, operated by the U.S. Department of Defense. GPS satellite signals can be used by anyone free of charge. At one time, the signals were modified so nongovernmental users could not obtain the greatest accuracy possible. This "selective availability" was discontinued by the federal government in 2000; however, for military purposes, "selective deniability" can still be used to degrade signals received by civilian GPS units in a war zone without affecting military GPS units.

The European Union has developed a plan to launch its own GPS system (the Galileo position system) by 2008. A GPS receiver performs location estimation by measuring the time difference of arrival of signals from four GPS satellites.

Prior to becoming available to the general pubic, GPS has been widely used in global navigation and in providing synchronization for cellular networks. Because the cost of a GPS receiver chipset continues to drop, GPS is generally considered the best choice for outdoor location sensing. For one thing, aside from working as a standalone mobile device, a GPS receiver chipset can be comfortably embedded into a cell phone or can be packed into a small expansion card. We believe that eventually every smart phone will have an onboard GPS chipset. In addition, many cars now have been equipped with a GPS navigation system.

A GPS satellite circles the Earth twice a day at an altitude of 20,200kin (about 12,600miles). Each GPS satellite has an atomic clock that keeps extremely precise time. To perform location calculations, a GPS receiver must receive signals from four GPS satellites.

The receiver then calculates a pseudorange, which is the time difference between its local clock and the time when a signal is sent from a satellite. By multiplying the speed of the radio signal and the pseudorange, the receiver can locate itself onto a sphere corresponding to each of the four satellites. The GPS receiver does not need to be equipped with a high-precision clock. As long as the receiver's local clock is stable in the short term, differences between the time points when GPS satellite signals are received can be measured quite accurately and will eventually yield the location of the GPS receiver.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

A GPS receiver can calculate its geographic location in terms of latitude, longitude, and altitude, although latitude and longitude are more likely to be used. An even more accurate system is Universal Transverse Mercator (UTM), a topographical rectangular grid consisting of 60 zones of 6 degrees of longitude; coordinates are expressed in meters east of the zone origin and north of the equator. For example, a location of N 40 ~ 05.425 W 075 ~ 07.035 can be converted into UTM to produce 18T E 490004 N 4437799.

The localization accuracy is measured by the distance deviation (e.g., 1 to 5 m). Another way to evaluate the accuracy of a GPS is precision, or the number of times a GPS receiver can give a claimed accuracy. Because a GPS receiver must obtain multiple signals from different satellites in order to calculate the first result, there is a significant delay called time to first fix (TFFF), which is commonly in the range of 20 to 40 sec. Network-assisted GPS (A-GPS) uses a large number of networked receivers to assist regular GPS receivers.

These networked receivers constantly collect GPS satellite signals, and a regular GPS receiver can request GPS data from these networked receivers instead of GPS satellites.

The global positioning system can generally fulfill the need for accurate positioning in outdoor settings and can provide worldwide coverage. Due to the nature of satellite signal reception, it does not perform well indoors.

In addition, because localization (which is highly computational intensive) is done on the GPS receiver, power consumption of a GPS chipset can be a problem that could impede the adoption of GPS-enabled smart phones.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

2.6 Apps

A mobile application, also referred to as a mobile app or simply an app, is a computer program or software application designed to run on a mobile device such as a phone, tablet, or watch. Apps were originally intended for productivity assistance such as email, calendar, and contact databases, but the public demand for apps caused rapid expansion into other areas such as mobile games, factory automation, GPS and location-based services, order-tracking, and ticket purchases, so that there are now millions of apps available.

Informative Apps and Enterprise Apps are explained in next chapter with example.

2.6.1 Informative Apps

The informative application is an application context in which the one and only goal is to provide information, like a news site, an online directory, a marketing site, or even a mobile commerce site, where the key task of the user is to read and understand and it is not necessary to interact (Figure 2-12). This isn't to say that you cannot include calls to action in the informative context—in fact, you should, but they should be based around what you can assume about your users in this context.

For example, remember that most mobile tasks are short and are often undertaken during brief idle periods. The user doesn't have much extra time and the task can be interrupted at any moment. In the case of a mobile news site, provide the user with the option to mark a page or story to be read later. With an online directory, allow the user to flag favorite entries. With a marketing site, allow users to enter the shortest possible contact information, like their phone number or email. And with a mobile commerce site, allow users to save items to a wishlist to review and purchase later.

The theme here is that although reading information is a simple task, it usually creates a complex chain of events that can be anticipated. With mobile applications, we want to avoid forcing the users to input too much information with their mobile devices, which is more difficult and takes more time than it would on another medium such as a desktop or laptop computers. Instead, we want to look for ways we can interconnect experiences, having users use the informative context to filter to the most desirable information when they have a moment, and allowing them to

interact with it later, when they have more time, from the medium of their choice.

Use informative applications when users need information to gain an understanding or perform a physical task, for applications that can be used in multimedia contexts such as desktop and mobile, for information-heavy applications, and for marketing or promotional applications.



Figure 2-12. An example informative application

Productivity Application Context

The productivity application context is used for content and services that are heavily task-based and meant to increase the users' sense of efficiency. With these types of applications, we can assume that the users are more committed to accomplishing a particular goal, like managing content such as messages, contacts, or media, but we should still assume that they are doing so during idle periods (Figure 2-13). Just because the application context is meant to make users more productive, we can't assume that they are able to make the same time commitment as they would in the desktop context.

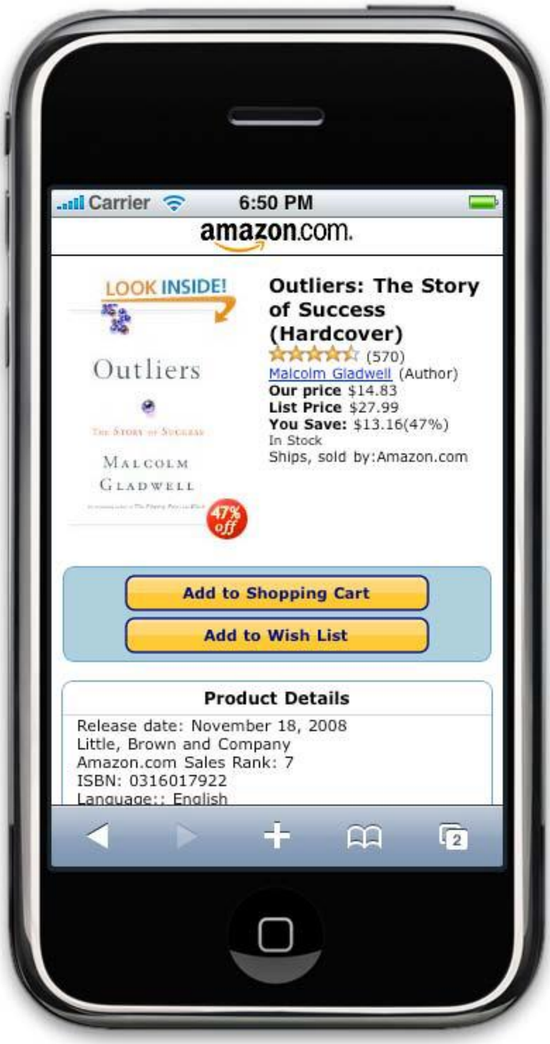


Figure 2-13. An example productivity application

Productivity applications are often very structured, presenting information in a defined hierarchy and often using the folder or group

metaphor to define a sense of order to the user. When designing these types of apps, we need to pay careful consideration to how the user thinks out the task. People have an uncanny ability to understand and recall complex hierarchies of tasks—for example, what they need to do first, second, and third in order for a particular solution to work. We take this for granted and in the desktop context often show the users the entire hierarchy visually. In the mobile context, we don’t have the screen real estate, and therefore need to help users find their way.

One method is to focus on prioritization of tasks; productivity applications typically include some method of direct or indirect prioritization. If we look at a mobile email client, we see that the app generally focuses around the inbox, which is the top-priority item, given that all new messages will come there first. All other folders are of a lower priority, as in order for messages to get there, we will have had to process them previously.

We can use this screen as a central focus point, assuming that users will spend the majority of their time there, and branching out onto other screens from this central spot. But we can’t forget other high-priority items, like the ability to send a new message or create a new item. This is a task that is typically included on every screen within an email app, and in the same position throughout to ensure that users always have quick access to create a new item.

The productivity context is one of the hardest application contexts to get right, so do yourself a favor and start simple. Start with one feature, treat it almost as if it were a single focus utility, and get it right before you move on to the next. Layer in your features one at a time until you feel like you have met the users’ goals, and stop the moment that it becomes it bit overwhelming to manage. You probably won’t be able to include every feature, so you will need to include only the ones that are most important to users, and lose everything else. Use the productivity application context for information-heavy applications where the user will need to manage content from a mobile device and for heavily structured, hierarchy-based tasks.

Immersive Full-Screen Applications

The final application context is an immersive full-screen application, like a game, a media player, or possibly even a single-screen utility. These applications are meant to consume the user’s focus, often doing so by filling the entire screen (Figure 2-14), and leaving no trace of the device user interface to distract the user. Again, the majority of mobile

engagement occurs when the user has idle periods of time; the immersive context is typical in most entertainment applications, one of the most popular mobile content areas.



Figure 2-14. An example of an immersive application

The most common use of the immersive context is obviously with a game, for which you want the user to focus on how to play the game. But this context can also be used with other contexts, presenting a full-screen view of content when the device orientation changes in many higher-end devices. For example, if we were making a localebased application, we could add a feature that changes the user experience to the immersive context, showing a full-screen map, or point-by-point directions, whenever

the device is held in landscape view. This is not a feature that many applications include, but I think it is worth considering.

Even with mobile web apps, many devices allow for detection of an orientation change. Typically, the app just scales to fill the page, actually breaking the intended user experience, but by adding the orientation-specific styles, the designer could create an immersive version of the application, presenting the app content in a more at-a-glance, friendly way, helpful for devices placed on automobile dashboards, or held in the hand to show others.

Use an immersive full-screen application for games, media players, and alternative views of another application context.

Application Context Matrix

Each of the application contexts in Table 2-2, comparing and contrasting their benefits to help you determine what is best for your application.

Table 2-2. Application context matrix

Application context Matrix	**User experience type**	**Task type**	**Task duration**	**Combine with**
Utility	At-a-glance	Information recall	Very short	Immersive
Locale	Location-based	Contextual information	Quick	Immersive
Informative	Content-based	Seek information	Quick	Locale
Productivity	Task-based	Content management	Long	Utility
Immersive	Full screen	Entertainment	Long	Utility, Locale

As you can see, mobile applications can run the gamut from intense experiences to simple tools. In some cases, they can switch back and forth between the two. Figure out which type of application is best for your users and in what context.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

2.6.2 Enterprise Apps

Mobile enterprise applications refer to a set of corporate applications that assist employee in performing business practice in many aspects, including fast data integration and decision making, real-time communication, and close control over goods and services. Mobile enterprise applications present a tremendous opportunity for enterprise application designers to take advantage of mobile computing in business operations. The goal of general enterprise applications is to enable business process automation in various scenarios.

Enterprise Application	Description	Mobile
Enterprise resource planning (ERP)	A highly customizable suite of applications for electronic processing of business operations; also referred to as an e-business package	ERP mobile client can be used to access a mobile web portal.
Customer relationship management (CRM)	A software solution that manages a customer database in association with sales, marketing, and product development information	CRM mobile client is used to access the CRM portal.
Supply-chain management (SCM)	A software solution that automates manufacturing, shipping, delivery, and purchasing of goods and services within and between companies	SCM can utilize RFID-based pallet and product tracking.
Collaboration	A set of communication tools that enables a variety of person-to-person communications and group collaboration methods, such as e-mail, group calendar and scheduler, enterprise instant messaging, voice over IP, and video conferencing	Collaboration includes mobile IM, mobile multimedia applications, wireless Internet applications, etc.
Knowledge management (KM)	A software solution to the management, analysis, and systematic presentation of information, work items, knowledge, and documents contributed by employees in a company	Acquisition and retrieval of knowledge can be done via a mobile device
Enterprise application integration (EAI)	The integration of various legacy and new enterprise applications within a company and between different companies	EAI should be transparent to mobile clients.

The above Table outlines widely used enterprise applications in the corporate world. Note that all enterprise applications must support both back-office (within the corporate site) and front-office operations (at field sites or customer sites).

Among these enterprise systems, ERR is generally the most sophisticated and requires intensive on-site customization and

optimization when it is deployed in a company, whereas CRM and KM may simply work off the shelf.

ERP essentially provides a data layer platform upon which a broad set of business processes can be interconnected and automated, such as bills of materials (BOMs), purchasing, accounts payable, manufacturing resource planning (MRP), sales and marketing, inventory management, etc. SCM is closely related to ERP with regard to sharing inventory and accounting information.

SCM is primarily concerned with improving the flow and efficiency of the supply chain, which can be broken down into five stages: supply chain planning, sourcing, manufacturing, delivery, and return. CRM and partner relationship management (PRM) are stand-alone applications focusing on maximizing the benefits of customer assets or partner assets.

Three types of business processes are involved in CRM: sales, which uses customer data for sales planning, sales analytics, and account management; marketing, which uses customer data to plan and manage campaigns and promotions; and service, which primarily deals with customer services. Customers can directly interface to CRM to perform self-guided operations. Obviously CRM and ERP share some functionalities as well as customer data. KM is not as common an enterprise application as ERP, SCM, and CRM.

All these systems can be exposed in some way to a collaboration platform that enables effective communication in a virtual working environment; yet, all these systems, including the collaboration platform and legacy enterprise systems, could possibly be integrated into a more strategic, enterprise-wide solution using some middleware. Between companies, web services are often used to enable interoperability.

The back end of enterprise applications always sits in a wired network, whereas the front end could be running on wired desktop computers or mobile devices within the enterprise network or on remote computers and mobile devices on the Internet via virtual private networks (VPNs). Enhancing these enterprise applications with mobile features could extend the reach of the application, thereby improving work productivity and reducing operational cost. The simplest form of mobile enterprise applications is supplying a mobile client that allows a mobile worker or a customer to access back-end systems anywhere, anytime.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

For each type of enterprise applications, a new set of mobile modules that enable augmented collaboration and intelligent business processing can be integrated into an enterprise system.

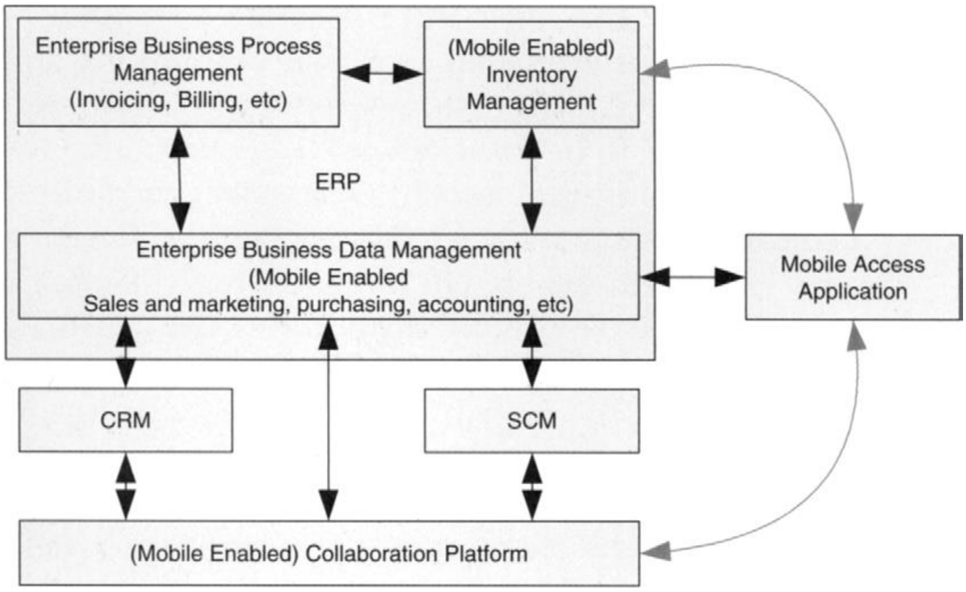


Figure 2-15 Mobile enterprise applications.

Figure 2-15 provides an overview of the enterprise application environment augmented with mobile technologies. Two types of applications for mobile wireless technologies could be used in an enterprise business environment. One is to build wireless networks and corresponding applications that directly engage in business processes such as manufacturing, floor planning, and inventory management, as part of the entire enterprise information and business processing system. Examples of this type of mobile application are wireless sensor networks in a manufacturing plant, location-based asset tracking in a hospital using wireless LAN or Bluetooth, and RFID-based inventory management in a warehouse.

The other type of application is to provide optimized mobile client portals on a mobile device for employees working in the field or at customer sites; for example, onsite field engineers can use mobile devices to directly access a backend database in real time to obtain job details and schedules and to communicate with other field workers on other sites. These two types of mobile applications essentially "mobilize" an enterprise system from the internal to the external.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Below is a summary of potential mobile applications for enterprise systems. These applications are by no means exhaustive, but they do illustrate the diverse ways mobile enterprise applications can help a company to conduct business:

1. Inventory management---Traditionally, inventory management has relied on manual scanning of a UPN (Universal Personal Number) code on pallets or products, which is time consuming and error prone. As RFID technology matures, low-cost RFID tags attached to pallets and products can be polled wirelessly by a nearby RFID reader. These RFID readers are furthered connected to server via a wired network, where raw tag data are mapped into detailed product information, aggregated, and sent to the central inventory system. A reader can also be attached to a mobile device to permit convenient point-and-check on the move. This scheme is actually a binary form of a proximity-based localization system. This may not suffice when accurate location of a product is needed, such as in a huge warehouse. Using some RFID tags placed at fixed locations as reference tags, it is possible to track the locations of product tags in real time. As mentioned earlier in the location-aware computing section, RFID can be combined with wireless LAN to provide real-time locating and monitoring (RTLM) for indoor asset management. Both ERP applications and SCM applications of a company or a retail store can be enhanced with RTLM to improve productivity (e.g., discovering shortages of materials, placing orders, confirming orders). Technical requirements of such systems include sufficient localization accuracy, long battery life of tags, seamless integration with other enterprise systems, and easy deployment and upgrade.
2. Onsite processing --- Field engineers, salespersons, and mobile workers at a remote site may take advantage of mobile client applications to access back-end enterprise systems in real-time. For example, by using a mobile device that wirelessly connects to a Job scheduling system, a field engineer can be dynamically assigned to a work item and can retrieve detailed information about the work item without going back to the office. While a phone call can be an option for quick and direct communication, it is generally difficult, if not impossible, to exchange detailed information such as numeric data and pictures in a phone call, not to mention the manpower of operators necessary to answer phone calls. When the work item is done, the engineer may update the status of the work item in the back-end database, and check out the next item. By using a

multifunction device such as a smart phone, the engineer can also send rich-media content of the work item to the back-end server, including pictures, audio description, instrumental measurements, and video clips. The procedure does not require any manual intervention on the server side. For salespersons and mobile workers who travel to a customer's site, a mobile client application running on a mobile device will allow them to conduct CRM-related business practices onsite, such as collecting and updating customer data, providing quick responses to customers' inquiries, checking product availability, generating real-time quotes for products or services, and onsite invoicing. Because of the enhanced quality of service offered by the salesforce, customer loyalty is likely to be improved. Another benefit of onsite processing is the increasing granularity of enterprise workflow, leading to agile and well-justified decision making. Onsite processing can be done with a single mobile portal, which is a unified mobile interface to back-end enterprise applications on a mobile device. The principle challenge to facilitating mobile-enhanced onsite processing is providing a range of sophisticated interoperable enterprise business processes on a mobile device of small form factor, unreliable wireless capability, and limited computing power. Obviously, a well-designed enterprise application integration (EAI) will make this a lot easier at the back end, thus the front end - the much simplified mobile access application on a mobile device - only has to interface with EAI rather than individual enterprise applications.

3. Onsite collaboration --- Field engineers, salespersons, and mobile workers may also use the wireless capability of a mobile device to communicate with colleagues back at the office or on other sites. Aside from e-mail, voice mail, and phone calls, a wide range of communication methods can be used on a mobile device, including IM, voice over IP, video streaming, and video conferencing. A major difference between these tools and consumer-oriented collaboration tools is that they are well suited for enterprise environment and can be integrated into other enterprise applications. For example, location-based presence service in mobile IM is particularly useful for a user who wants to find out the current work status of other colleagues. Imagine an enterprise IM application running on a mobile device that shows the geographic location of colleagues and their work status. This would improve job scheduling as well as work item tracking. Field engineers,

mobile workers, and salespersons in a conversation can quickly refer to the same piece of business data retrieved from a backend enterprise system, thereby improving communication efficiency and work productivity. Besides, enterprise security is a major difference that separates mobile enterprise collaboration tools with the counterparts for the mass public. In order to deal with the peculiarities of mobile enterprise applications, a company usually imposes strict policies for the establishment of secured computing infrastructure as well as user usage. An example of mobile security in the enterprise is the discovery of rogue wireless access points. A rogue wireless access point is an access point installed without authorization and thus does not conform to enterprise network security policies. A rogue wireless access point could be a serious security problem because it exposes a supposedly well-protected enterprise network. Enterprises often use wireless intrusion detection systems along with positioning techniques to approximate the location of rogue access points and then use a mobile device to pinpoint them. Some wireless intrusion detection systems even combine location with authentication to deny access to stations that lie outside a predefined physical perimeter, such as AirTight SpectraGuard (http://www.airtightnetworks.net/) and Newbury Watchdog (http://www.newburynetworks.com).

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

2.7 Check your progress Questions (Multiple Choice Questions)

1. Which one of the following is not a categories of mobile application medium types?
   a) Mobile Websites b) Games
   c) Location Based Services d) Android

2. Which one of the following is the limitation of characters in.SMS?
   a) 160 b) 170
   c) 180 d) 190

3.The mobile websites are ______.
   a) easy to create, maintain, and publish.
   b) difficult to support across multiple devices.
   c) All of the above d) None of the above

4. Find out the odd one
   a) Game b) Windows
   c) Android d) IOS

5. Mobile Apps are primary split into three categories. They are ______.
   a) Native, Web, and Hybrid
   b) Licensed, customized, and open source.
   c) Hybrid, exportable and open source.
   a) Customized, prioritized and Web.

6. Calculator App is an (a) ________ App.
   a) Informative b) LBS
   c) Enterprise d) Utility

2.8 Answer to Check your progress Questions

1. d) Android
2. a) 160
3. c) All of the above
4. a) Game
5. a) Native, Web, and Hybrid
6. d) Utility

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

2.9 Summary

Mobile Websites, Mobile Web Widgets, Short message service(SMS), Native Applications, Games, Utility Apps, Location Based Services(LBS), Informative Apps, and Enterprise Apps are categories of mobile medium applications.

SMS (short message service) is a text messaging service component of most of the mobile device. SMS enablement allows individuals to send an SMS message to a business phone number (traditional landline) and receive a SMS in return. Providing customers with the ability to text to a phone number allows organizations to offer new services that deliver value.

The mobile web refers to browser-based World Wide Web services accessed from handheld mobile devices, such as smartphones or feature phones, through a mobile or other wireless network. However, the web is now more accessible by portable and wireless devices.

Mobile game is a game played on a feature phone, smartphone, tablet, smartwatch, PDA, portable media. In 1997, Nokia launched the very successful Snake. Snake (and its variants), that was preinstalled in most mobile devices manufactured by Nokia, has since become one of the most played games and is found on more than 350 million devices worldwide.

Calculator is the best example for utility app. It is used to compute standard calculations and scientific calculations. Assistive healthcare systems, recommending social events in a city, requesting the nearest ATM, restaurant or a retail store, locating people on a map displayed on the mobile phone, receiving alerts, such as notification of a sale on gas or warning of a traffic jam, and location-based mobile advertising are some of the applications of Location based system services.

Informative apps are very use full in Educational online learning system. Enterprise apps target a wide range of industries and can integrate with office, purchase, sales and customer service processes.

2.10 Keywords

Mobile Categories, Mobile Applications, SMS, Mobile Websites, Mobile Web Widgets, Native Applications, Platforms, Games, Applications, Services, Location Based Services, Informative apps, Enterprise Apps.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

2.11 Self Assessment Questions

Short Answer Questions

1. Define the term ‘mobile devices profiles’.
2. Write the categories of mobile applications.
3. Define short message service (SMS)
4. Write the advantages of mobile websites
5. List out some native mobile applications.
6. Define Location Based Services(LBS).
7. Why Informative Apps are very important in learning?
8. How Enterprise app interconnects all areas in an organization?

Long Answer Questions

1. Explain the following concepts in detail
    a) Native apps
    b) Hybrid apps
    c) Web apps
    d) Utility apps
2. Discuss the short message service application and its usage in detail.
3. Explain the usage of mobile web widgets.
4. Why native apps are very essential? Give valid reasons.
5. Discuss game apps in detail.
6. Write the working principles of Location Based Services(LBS).
7. How the Informative Apps will improve the learning process?
8. Explain Enterprise Apps in detail.
9. Explain the following concepts in detail
    a) Enterprise Apps
    b) Informative Apps
10. Compare and contrast Enterprise apps and Informative apps.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

2.12 Further readings

1. Mobile Design and Development by Brian Fling, O'Reilly Media, Inc 2009

2. J2ME: The Complete Reference, James Keogh, Tata McGrawHill 2003

3. . Smart Phone and Next-Generation Mobile Computing by Pei Zheng and Lionel Ni, Elseveir, 2006.

4. Beginning Android by Mark L. Murphy , Apress 2009

5. H. Lashkari, M. Moradhaseli, Mobile operating systems and programming: mobile communications, VDM Verlag Dr. Muller, 2011

6. http://opengardensblog.futuretext.com/archives/2008/06/iphone_vs_symbi_1.html

7. http://www.isecpartners.com/files/iSEC_Securing_Android_Apps.pdf

8. http://www.ibm.com/developerworks/architecture/library/wi-arch23.html

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

3. MOBILE INFORMATION ARCHITECTURE

3.1 Contents of the unit

3.2 Introduction

Information architecture (also known as IA), is the foundation of mobile product. A well-engineered product with good visual design can still fail because of poor information architecture. The truly successful mobile products always have a well thought- out information architecture.

From a simple mobile website to an iPhone application, the mobile information architecture defines not just how your information will be structured, but also how people will interact with it. This is made especially tricky when you consider that different devices have different capabilities and therefore different interaction models.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Take the way people interact with their devices: for example, a touch device on which the user literally points and clicks, or a more basic device on which the user uses the directional pad to navigate to the desired location

3.3 Objectives

- To understand Mobile Information Architecture
- To learn about prototypes
- To learn Mobile design and its tools

3.4 Mobile Information Architecture

The organization of data within an informational space. In other words, how the user will get to information or perform tasks within a website or application.

Interaction design

The design of how the user can participate with the information present, either in a direct or indirect way, meaning how the user will interact with the website of application to create a more meaningful experience and accomplish her goals.

Information design

The visual layout of information or how the user will assess meaning and direction given the information presented to him.

Navigation design

The words used to describe information spaces; the labels or triggers used to tell the users what something is and to establish the expectation of what they will find.

Interface design

The design of the visual paradigms used to create action or understanding.

Mobile Information Architecture

Although information architecture has become a common discipline in the web industry, unfortunately, the mobile industry—like software—has only a handful of specialized mobile information architects. Although mobile information architecture is hardly a discipline in its own

right, it certainly ought to be. This is not because it is so dissimilar from its desktop cousin, but because of context, added technical constraints, and needing to display on a smaller screen as much information as we would on a desktop.

Keeping It Simple

When thinking about your mobile information architecture, you want to keep it as simple as possible.

Support your defined goals

If something doesn't support the defined goals, lose it. Go back to your user goals and needs, and identify the tasks that map to them. Find those needs and fill them.

Ask yourself: what need does my application fill? What are people trying to do here? What is their primary goal? Once you understand that, it is a simple process of reverse engineering the path from where they want to be to where they are starting. Cut out everything else—your site or application doesn't need it.

For example, to get some news and information on a mobile device, you need to first ask what the goal is. What is the need you are trying to fill? Then you need to apply context. Where are your users? What are they doing? Are they waiting for the bus? Do they have only a minute to spare? Or, do they have five minutes to spare? With these answers, you get your information architecture.

Clear, simple labels

Good trigger labels, the words we use to describe each link or action, are crucial in Mobile. Words like "products" or "services" aren't good trigger labels. They don't tell us anything about that content or what we can expect. Now, I would argue that good trigger labels are crucial in the Web as well, that we've become lazy and we assume so much about the user that we ignore the use of good trigger labels.

Users have a much higher threshold of pain when clicking about on a desktop site or application, hunting and pecking for tasty morsels. Mobile performs short, to-the-point, get-it-quick, and get-out types of tasks. What is convenient on the desktop might be a deal breaker on mobile. Keep all your labels short and descriptive, and never try to be clever with the words you use to evoke action.

3.4.1 Site Maps

Site maps are a classic information architecture deliverable. They visually represent the relationship of content to other content and provide a map for how the user will travel through the informational space, as shown in Figure 3-1.

Mobile site maps aren't that dissimilar from site maps we might use on the Web. But there are a few tips specific to mobile that we want to consider.

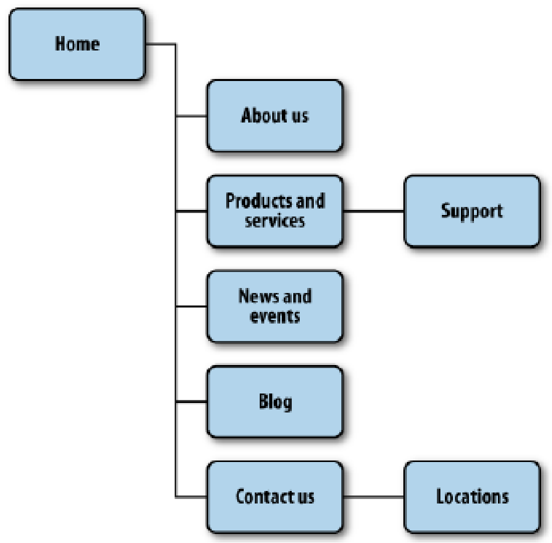


Figure 3-1 An example mobile site map

Limit opportunities for mistakes

In the mobile context, tasks are short and users have limited time to perform them. And with mobile websites, we can't assume that the users have access to a reliable broadband connection that allows them to quickly go back to the previous page. In addition, the users more often than not have to pay for each page view in data charges. So not only do they pay cash for viewing the wrong page by mistake, they pay to again download the page they started from: we can't assume that pages will be cached properly. Therefore it is advised to limit users' options.

Confirm the path by teasing content

After the users have selected a path, it isn't always clear whether they are getting to where they need to be. Information-heavy sites and applications often employ nested or drill-down architectures, forcing the

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.
user to select category after category to get to their target. To reduce risking the user's time and money, we want to make sure we present enough information for the user to wade through our information architecture successfully.

On the Web, we take these risks very lightly, but with mobile, we must give our users a helping hand. We do this by teasing content within each category— that is, providing at least one content item per category.

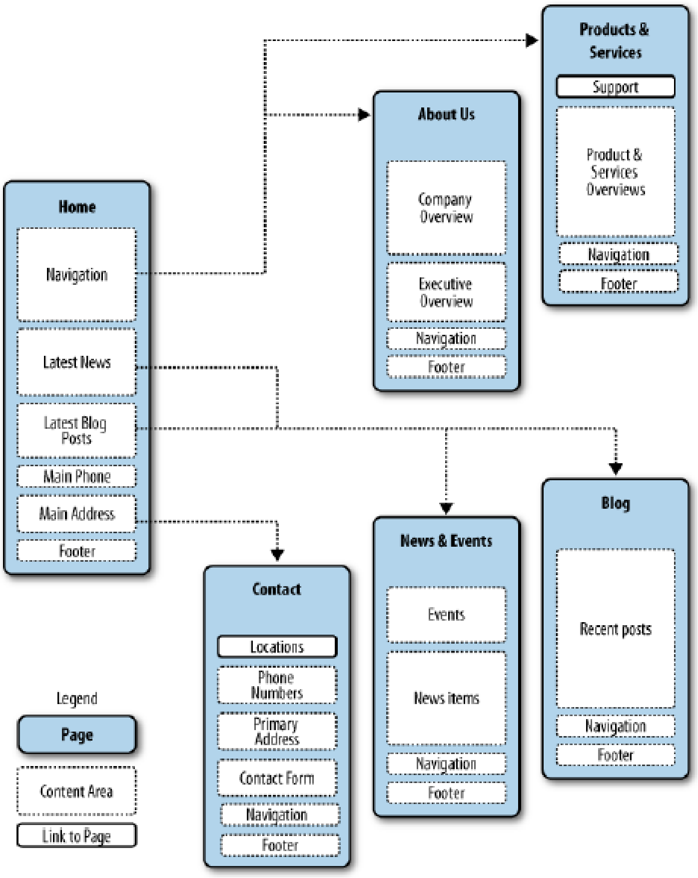


Figure 3-2 Teasing content to confirm the user's expectations of the content within

The challenge with ringtone sites is you have a lot of items, grouped by artist, album, genre, and so on. The user starts with a goal like "We want a new ringtone" and finds an item that suits his taste within a catalog of tens of thousands of items. In order to make sense of a vast inventory of content, we have to group, subgroup, and sometimes even subgroup again, creating a drill-down path for the user to browse.

In Figure 3-2, you can see in a constrained screen that teasing the first few items of the page provides the user with a much more intuitive interface, immediately indicating what type of content the user can expect.

We immediately saw that users were finding content more quickly, driving up our sales.

3.4.2 Clickstreams

Clickstream is a term used for showing the behavior on websites, displaying the order in which users travel through a site's information architecture, usually based on data gathered from server logs. Clickstreams are usually historical, used to see the flaws in your information architecture, typically using heat-mapping or simple percentages to show where your users are going. I've always found them to be a useful tool for rearchitecting large websites.

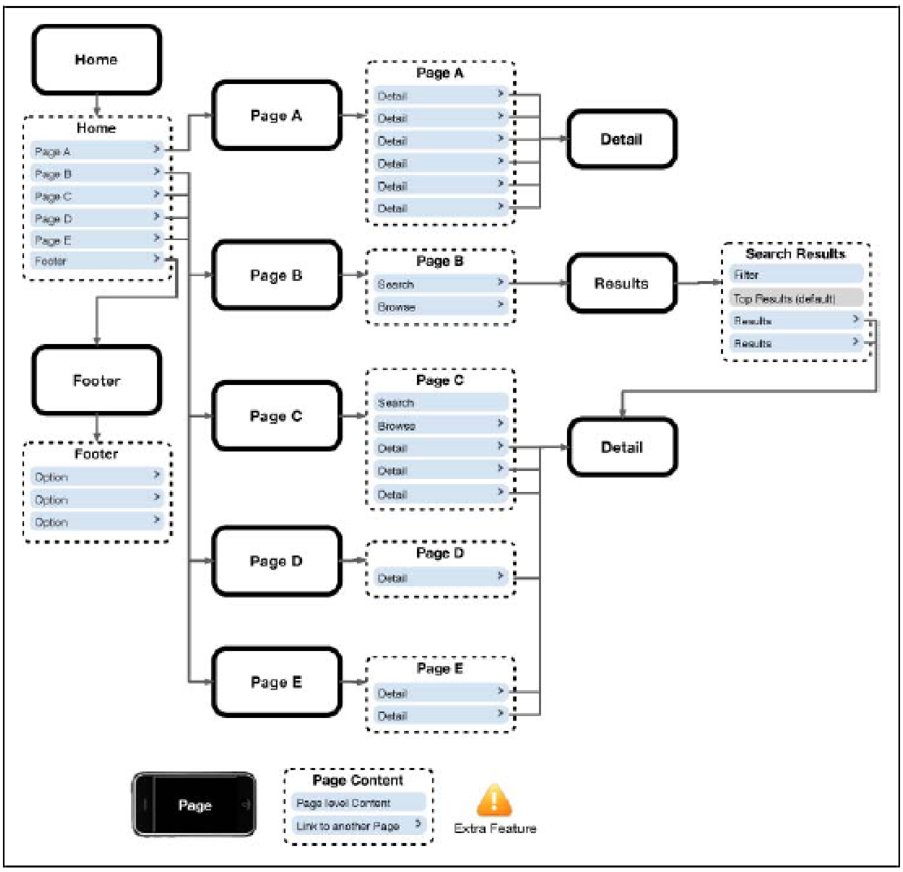


Figure 3-3 An example clickstream for an iPhone web application

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

However, information architecture in mobile is more like software than it is the Web, meaning that you create clickstreams in the beginning, not the end. This maps the ideal path the user will take to perform common tasks. Being able to visually lay out the path users will take gives you a holistic or bird's-eye view of your mobile information architecture, just as a road map does. When you can see all the paths next to each other and take a step back, you start to see shortcuts and how you can get users to their goal faster or easier, as shown in Figure 3-3.

Now the business analyst in you is probably saying, "Just create user or process flows," such as the esoteric diagram shown in Figure 3-4, which is made up of boxes and diamonds that look more like circuit board diagrams than an information architecture.

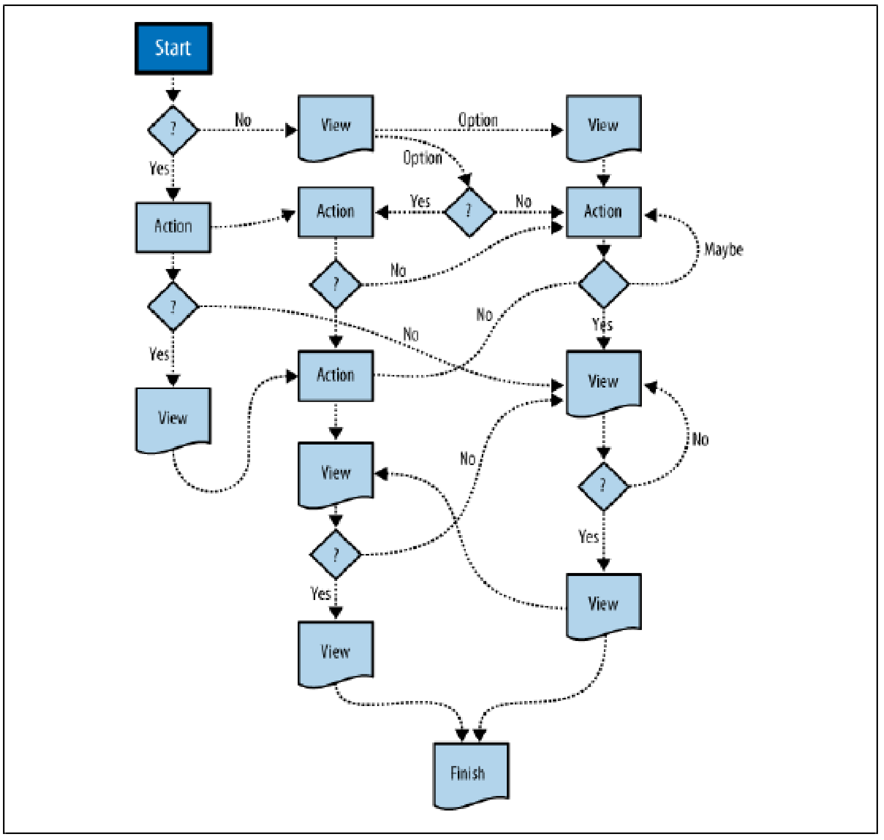


Figure 3-4 An example process flow diagram

A good architect's job is to create a map of user goals, not map out every technical contingency or edge case. Too often, process flows go down a slippery slope of adding every project requirement, bogging down the user experience with unnecessary distractions, rather than focusing on streamlining the experience. Remember, in mobile, our job is to keep it as

simple as possible. We need to have an unwavering focus on defining an excellent user experience first and foremost. Anything that distracts us from that goal is just a distraction.



Figure 3-5 An example of an iPhone web application wireframe, intended to be low fidelity to prevent confusion of visual design concepts with information design concepts

3.4.3 Wireframes

Wireframes are a way to lay out information on the page, also referred to as *information design*. Site maps show how our content is organized in our informational space; wireframes show how the user will directly interact with it. Wireframes are like the peanut butter to the site map jelly in our information architecture sandwich. Wireframes like the one in Figure 3-4 serve to make our information space tangible and useful.

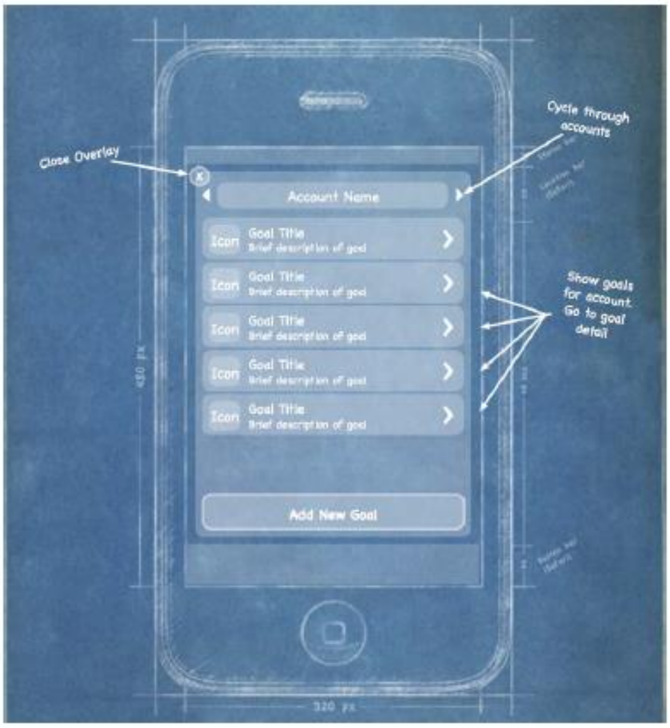


Figure 3-6 Using annotations to indicate the desired interactions of the site or application

But the purpose of wireframes is not just to provide a visual for our site map; they also serve to separate layout from visual design, defining how the user will interact with the experience. How do we lay out our navigation? What visual or interaction metaphors will we use to evoke action? What are the best ways to communicate and show information in the assumed context of the user? These questions and many more are answered with wireframes.

The "in-place" interactions, or areas where the user can interact with an element without leaving the page. This can be done with Ajax or a little show/hide JavaScript. These interactions can include copious amounts of annotation, describing each content area in as much length as you can fit in the margins of the page, as shown in Figure 3-6.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

3.4.4 Prototyping

Wireframes lack the capability to communicate more complex, often in-place, interactions of mobile experiences. This is where prototypes come in.

Prototypes might sound like a scary (or costly) step in the process. Some view them as redundant or too time-consuming, preferring to jump in and start coding things. But as with wireframes, some sort of prototype has saved both time and money. The following sections discuss some ways to do some simple and fast mobile prototyping.

Paper prototypes

The most basic level we have is paper prototyping: taking our printed-out wireframes or even drawings of our interface, like the one shown in Figure 3-7, and putting them in front of people.

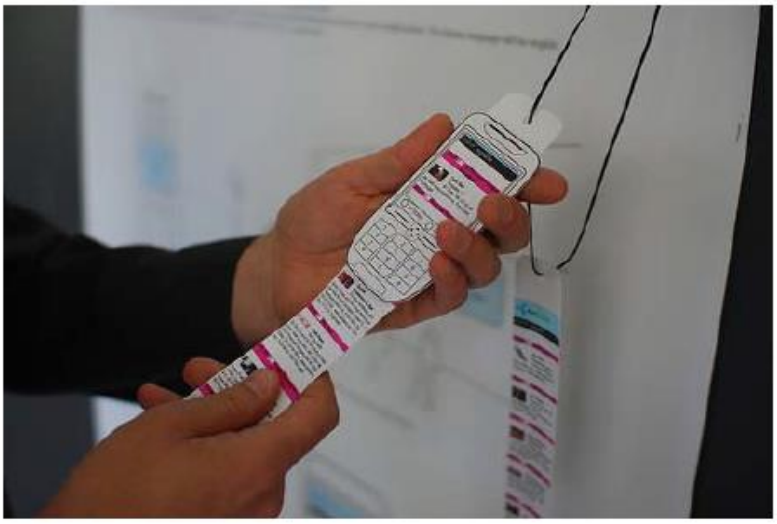


Figure 3-7 A paper prototype, while the interaction is nothing more than drawings on note cards



Figure 3-8 A touch interface paper prototype next to its smaller sibling

Create a basic script of tasks (hopefully based on either context or user input) and ask users to perform them, pointing to what they would do. You act as the device, changing the screens for them. For paper prototypes, use small blank note cards, and for lower-end devices, use business card- sized paper (Figure 3-8).

Context prototype

The next step is creating a context prototype (Figure 3-9). Take a higher-end device that enables you to load full-screen images on it. Take your wireframes or sketches and load them onto the device, sized to fill the device screen. Leave the office. Go for a walk down to your nearest cafe. Or get on a bus or a train. As you are traveling about, pull out your device and start looking your interface in the various contexts you find yourself currently in.

Pay particular attention to what you are thinking and your physical behavior while you are using your interface and then write it down. If you are brave and don’t have strict nondisclosure issues, ask the people around you to use it, too. I wouldn’t bother with timing interactions or sessions, but try to keep an eye on a clock to determine how long the average session is.



Figure 3-9 An example of a context prototype or taking images loaded onto a device and testing them in the mobile context

HTML prototypes

The third step is creating a lightweight, semi functional static prototype using XHTML,CSS, and JavaScript, if available. This is a prototype that you can actually load onto a device and produce the nearest experience to the final product, but with static dummy content and data (Figure 3-10). It takes a little extra time, but it is worth the effort.

With a static XHTML prototype, you use all the device metaphors of navigation, you see how much content will really be displayed on screen (it is always less than you expect), and you have to deal with slow load times and network latency. In short, you will feel the same pains your user will go through.

Whatever route you wish to take, building a mobile prototype takes you one very big leap forward to creating a better mobile experience.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.



Figure 3-10 An XHTML prototype that you can actually interact with on real mobile devices

3.4.5 Architecture

Mobile app architecture is a set of techniques and patterns used to develop fully structured mobile applications based on industry and vendor specific standards. While formulating the app architecture, the procedures that work on the wireless mobile device like smartphones and tablets are also taken into consideration.

Different Information Architecture for Different Devices

Depending on which devices you need to support, mobile wireframes can range from the very basic to the complex. On the higher-end devices with larger screens, we might be inclined to add more interactions, buttons, and other clutter to the screen, but this would be a mistake. Just because the user might have a more advanced phone, that doesn't mean that he is giving you license to pack his screen with as much information as you can muster.

The motivations, goals, and how users will interact with a mobile experience are the same at the low end as they are on a high-end device. For the latter, you just have better tools to express the content. You can learn a lot from designing for the lower end first.

The greatest challenge in creating valuable experiences knows when to lose what you don’t need. You don’t have a choice on lower-end devices—it must be simple.

When designing for both, it is best to try and to keep your information architecture as close to each other as possible without sacrificing the user experience. They say that simple design is the hardest design, and this principle certainly is true when designing information architecture for mobile devices.

The Design Myth

A little secret about interactive design is that people don’t respond to the visual aesthetic as much as you might think. What colors you use, whether you use square or rounded corners, or, gradients or flat backgrounds, helps build first impressions, but it doesn’t do too much to improve the user’s experience.Users appreciate good design, but they are quickly indifferent about the visual aesthetic and move almost immediately to the layout (information design), what things are called (taxonomy), the findability of content, and how intuitive it is to perform tasks. These are all facets of information architecture.



Figure 3-11 Comparing visual design to information design of the iPhone application Tweetie

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Just look at one of the top-selling iPhone Twitter applications, Tweetie, shown in Figure 3-11. Many consider Tweetie to be a "well-designed" application, but because it is built from the same API as all other iPhone applications, at first glance there is little that is actually visually distinctive between this and other applications. What makes this application "well designed" is how the content is applied to the context of the user—in other words, the mobile information architecture.

In this example, the information design uses common layout metaphors, highlighted on the righthand side of Figure 3-11 to provide the user with familiar placement of common tasks, allowing the user to perform repetitive tasks common with most Twitter applications. The point is great information design is often mistaken for great visual design.

Most non–information architects almost always do information architecture in some form or another; often, they don't even know they are doing it. They might do a few wireframes, or maybe a site map. Sometimes designers will jump in and incorporate information architecture deliverables directly into their designs. By not focusing on the information architecture exclusively from the start, you risk confusing your disciplines, your deliverables, and ultimately your direction. The more time you spend focusing on just your information architecture, the faster and less costly your project will be.

3.5 Mobile Design

When building mobile experiences, it is impossible to create a great experience without three ingredients: context, information architecture, and visual design. The visual design of your experience is the direct representation of everything underneath; it is the first impression the user will have. A great design gives the user high expectations of your site or application; a poor design leads to lower expectations.

Users' expectations translate to value and trust. In the mobile space, where content is often "free" (they still need to pay for data charges), users often have low expectations, due to the limitations of the canvas. Confusing site structures and slow download speeds reinforce those initial low expectations. In the downloadable application space, where application design can be much more robust, users must purchase applications almost sight unseen. For example, they may see just a small screenshot of your application or game. But if the application doesn't meet

the higher expectations of the design, your application downloads will drop like a stone. The design, that first impression, determines right from the start if the user will spend five seconds, five minutes, or five hours with your product.

This leads us to the most significant challenge in mobile design: creativity. You can’t always be as creative as you want to be. Many devices just can’t support complex designs for every channel; for example, on many lower-end devices, the mobile web experience may just be a list of links. But every device has the capability to create a best in-device experience; it just depends on how you take advantage of the application medium and context that you plan to use.

On computers, there is a strategy called “lowest common denominator”: in order to reach the widest possible number of platforms, you create a product that works on the most common architectural components on all platforms (see Figure 3-12). Well, in computers, where you may have under a dozen different platforms, this is a great concept, but in mobile development, where you might be dealing with hundreds of different devices, it becomes a necessity.



Figure 3-12 A lowest-common-denominator design

Typically, mobile design starts with the lowest common denominator. As a designer, you ask yourself, "How do I visually express this content across the most possible devices?" You start with the most basic of designs, catering to the limitations of the device. You try to pepper in some nice-looking elements until you've reached the extent that the device platform can tolerate. You are left with a Frankenstein-like design that only your mother could love.

3.5.1 Interpreting Design

A 25-year veteran graphic designer spent creating print designs and advertisements, defining precisely what each design would be, down to the picas and points. His method of design meant creating a vision for how to communicate information or ideas in his head, and then duplicating that on the printed page. In his mind, it wasn't right unless it was exactly like his original vision.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

He was offered with website. He spent months obsessing about exactly how his site looked. The method of communicating information was to structure it with design, use design elements to enhance the information, and enable the user to interpret it. It is important to remember that every experience is unique. That experience depends on the user's screen size, web browser, text settings, the speed of his computer, and his connection to the Internet. There are simply too many variables for us to try to "control" the design completely.

This frustrated the veteran graphic designer. He could not look at interpreted designs without trying to precisely position every element on the page. And if the browser window is resized, he would get angry.

The great thing about design—it is completely subjective, giving designers plenty of things to argue about. We just had two very different ways of using design to express information, based on the fact that we came from different media. He wanted it to treat the design precisely, to recreate his vision exactly.

We wanted it to be flexible, catering to the unknown variables of the medium. The reality is that we probably should have met someplace in the middle.

Mobile design isn't that different. Precise designs might look better, but they can be brutal to implement. More flexible designs might not be much to look at, but they work for the most users, or the lowest common denominator. But more than that, our backgrounds and our training can actually get in the way of creating the best design for the medium. We like to apply the same rules to whatever the problem in front of us might be. In mobile design, you interpret what you know about good design and translate it to this new medium that is both technologically precise and at times incredibly unforgiving, and you provide the design with the flexibility to present information the way you envision on a number of different devices.

In the end, the graphic designer and I scrapped the work, and he provided me his pica perfect designs as giant images, which I turned into a series of massive image maps.

The Mobile Design Tent-Pole

In Hollywood, executives like to use the term "tent-pole" to describe their movies and television shows. The term has dual meanings: one is business, and the other creative.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

The business goal of a tent-pole production is to support or prop up the losses from other productions. However, to create a tent-pole production, the creators involved must make an artistic work that they know will appeal to the largest possible audience, providing something for everyone. You probably know tent-pole movies as “blockbusters”; in television, they are known as “anchor” shows.

Trying to reach the widest possible audience poses a problem. Hollywood is learning with great pains that with so many entertainment options and with today’s audience being so hard to reach through traditional advertising channels, tent-pole productions are failing. As the number of social niches increases, it becomes difficult to satisfy the specific tastes of each social group. What one group finds hysterically funny, several other groups might find offensive. Today, tent-pole productions often come off as bland and half-hearted, failing to appease anyone.

One of the most interesting examples of how the tide turned in entertainment is with the animated films of Disney versus those of Pixar. For years, Disney produced tentpole family fare quite successfully. But as competition increased, notably from Pixar, Disney films would spend millions to create stale and dated films, losing audiences and revenue. Meanwhile, Pixar found that their movies could be successful by avoiding the traditional storytelling formats of animated film, which Disney essentially defined. Instead, Pixar based their stories around specific emotional themes and was able to connect with audiences of all ages, in multiple cultures and across multiple niches.

In 2006, Disney acquired Pixar, making its top executives the new leaders of all Disney creative projects. Disney realized that it needed to be more Pixar and less Disney in order to grow and adapt to today’s changing audiences and niches. This is something that Pixar was doing correctly.

Back in the world of mobile design, the de facto strategy is to create tent-pole products.

Like the old days of Disney, the strategy is to sink millions into creating tent-pole products, or products that support the largest number of devices that no one will ever use. They are creatively stale, they lack inspiration, and they simply exist with no meaningful purpose to the user. They make the same mistake Disney made, thinking that it could simply put something on the market that might not be the best quality, but because it carried the Disney name, people would buy it.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

To have a successful mobile design, you have to think like Pixar. Find that emotional connection, that fundamental need that serves many audiences, many cultures, and many niches and design experiences. Too often, designers simply echo the visual trends of the day, mimicking the inspiration of other. But with mobile design, once you find that essential thing, that chewy nougat we call "context" that lives at the center of your product, then you will find ample inspiration of your own to start creating designs that translate not only across multiple devices, but across multiple media.

Sure, there are countless examples of poorly designed mobile products that are considered a success. You only need to look as far as the nearest mobile app store to find them. This is because of the sight unseen nature of mobile commerce. Traditionally, you couldn't demo—or in some cases even see—screenshots of a game or mobile application before you bought it. You simply had to purchase it and find out if it was any good.
Apple's App Store quickly changed that. You can clearly see that the best-selling games and applications for the iPhone are the ones with the best designs (Figure 3-13).



Figure 3-13 The app icon design greatly influences the user's expectation of quality

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Users look at multiple screenshots (Figure 3-14), read the user reviews, and judge the product based on the quality of its icon and of the screenshots before they buy.



Figure 3-14 Users are able to determine the quality of the app, largely influenced by the design, before they make a purchase

The Apple App Store is proving everyday that mobile design doesn’t have to start with tent-pole lowest-common-denominator products—it can instead start with providing the best possible experience and tailoring that experience to the market that wants it most.

Designing for the Best Possible Experience

When the first iPhone came out, I got in a lot of trouble from my web and mobile peers for publicly saying, “The iPhone is the only mobile device that matters right now.” They would argue, “What about ABC or XYZ platforms?” My response was that those are important, but the iPhone provides the best possible experience and that is where consumers will go. Since those days, we’ve seen the iPhone shatter just about every record in mobile devices, becoming one of the best-selling phones ever and one of the most used mobile browsers in the world—two-thirds of mobile browsing in the U.S. comes from an iPhone or an iPod touch, not

to mention that more than a billion mobile applications have been sold for these devices in under a year.

Recently, I was speaking at a conference where I ran into one of my peers, who questioned my premise that the iPhone was the most important device in mobile. He came up to me, and the first thing he said was, "I remember you telling me ages ago that the iPhone is the only device that mattered, and I didn't believe you. And here we are today focusing our business on the iPhone." It was an odd (and rare) reverse I-told-you-so moment. Here was this seasoned mobile guy telling me that his instincts had been wrong and my instincts had been right. I thought it must have been hard for him to go against his instincts and shift not just his thinking but his entire business toward supporting one popular device. The lesson here is that although it may defy your business instincts to focus your product on just one device, in mobile development, the risks and costs of creating that tentpole product are just too high. This lesson is so easily seen through bad or just plain uninspired mobile design. Asking creative people to create uninspiring work is a fast track to mediocrity.

Here is a design solution: design for the best possible experience. Actually, don't just design for it: focus on creating the best possible experience with unwavering passion and commitment. Iterate, tweak, and fine-tune until you get it right. Anything less is simply unacceptable. Do not get hindered by the constraints of the technology. Phrases like "lowest common denominator" cannot be part of the designer's vocabulary. Your design—no, your work of art—should serve as the shining example of what the experience should be, not what it can be. Trying to create a mobile design in the context of the device constraints isn't where you start; it is where you should end.

I think one of the greatest mistakes we in the mobile community make is being unwilling to or feeling incapable of thinking forward. The tendency to frame solutions in the past (past devices, past standards) applies only to those low-quality, something-foreveryone- but-getting-nothing tent-pole products. Great designs are not unlike great leaps forward in innovation. They come from shedding the baggage regarding how things are done and focus on giving people what they want or what they need.

3.5.2 The Elements of Mobile Design

The good design requires three abilities: the first is a natural gift for being able to see visually how something should look that produces a desired emotion with the target audience. The second is the ability to

manifest that vision into something for others to see, use, or participate in. The third is to know how to utilize the medium to achieve your design goals.

The six elements of mobile design that you need to consider, starting with the context and layering in visual elements or laying out content to achieve the design goal. Then, you need to understand how to use the specific tools to create mobile design, and finally, you need to understand the specific design considerations of the mobile medium.

Context

The context is core to the mobile experience. As the designer, it is your job to make sure that the user can figure out how to address context using your app. Make sure you do your homework to answer the following questions:

- Who are the users? What do you know about them? What type of behavior can you assume or predict about the users?
- What is happening? What are the circumstances in which the users will best absorb the content you intend to present?
- When will they interact? Are they at home and have large amounts of time? Are they at work where they have short periods of time? Will they have idle periods of time while waiting for a train, for example?
- Where are the users? Are they in a public space or a private space? Are they inside or outside? Is it day or is it night?
- Why will they use your app? What value will they gain from your content or services in their present situation?
- How are they using their mobile device? Is it held in their hand or in their pocket? How are they holding it? Open or closed? Portrait or landscape?

The answers to these questions will greatly affect the course of your design. Treat these questions as a checklist to your design from start to finish. They can provide not only great inspiration for design challenges, but justification for your design decisions later.

Message

Another design element is your message, or what you are trying to say about your site or application visually. One might also call it the “branding,” although I see branding and messaging as two different things. Your message is the overall mental impression you create explicitly through visual design. I like to think of it as the holistic or at times instinctual reaction someone will have to your design. If you take a step

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.
back, and look at a design from a distance, what is your impression? Or conversely, look at a design for 30 seconds, and then put it down. What words would you use to describe the experience?

Branding shouldn't be confused with messaging. Branding is the impression your company name and logo gives—essentially, your reputation. Branding serves to reinforce the message with authority, not deliver it. In mobile, the opportunities for branding are limited, but the need for messaging is great. With such limited real estate, the users don't care about your brand, but they will care about the messaging, asking themselves questions like, "What can this do for me?" or "Why is this important to me?"

Your approach to the design will define that message and create expectations. A sparse, minimalist design with lots of whitespace will tell the user to expect a focus on content. A "heavy" design with use of dark colors and lots of graphics will tell the user to expect something more immersive.

For example, hold the book away from you and look at each of the designs in Figure 3-15; try not to focus too heavily on the content. What do each of these designs "say" to you?



Figure 3-15 What is the message for each of these designs?

Which of the following designs provide a message? What do they say to you?

Yahoo!

Yahoo! sort of delivers a message. This app provides a clean interface, putting a focus on search and location, using color to separate it from the news content. But I'm not exactly sure what it is saying. Words you might use to describe the message are crisp, clean, and sharp.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

ESPN

The ESPN site clearly is missing a message. It is heavily text-based, trying to put a lot of content above the fold, but doesn't exactly deliver a message of any kind. If you took out the ESPN logo, you likely would have indifferent expectations of this site; it could be about anything, as the design doesn't help set expectations for the user in any way. Words you might use to describe the message: bold, cluttered, and content-heavy.

Disney

Disney creates a message with its design. It gives you a lot to look at—probably too much—but it clearly tries to say that the company is about characters for a younger audience. Words you might use to describe the message: bold, busy, and disorienting.

Wikipedia

The Wikipedia design clearly establishes a message. With a prominent search and text-heavy layout featuring an article, you know what you are getting with this design. Words you might use to describe the message: clean, minimal, and text-heavy.

Amazon

Amazon sort of creates a message. Although there are some wasted opportunities above the fold with the odd ad placement, you can see that it is mostly about products (which is improved even more if you scroll down). Words you might use to describe the message: minimal but messy, product-heavy, and disorienting. The words you might use to describe these designs might be completely different than mine—thus the beauty and the curse of visual design. The important thing isn't my opinion—it is the opinion of your user. Does the design convey the right message to your user in the right context? If you aren't sure, it might be a good time to find out.

Look and Feel

The concept of "look and feel" is an odd one, being subjective and hard to define. Typically, look and feel is used to describe appearance, as in "I want a clean look and feel" or "I want a usable look and feel." The problem is: as a mobile designer, what does it mean? And how is that different than messaging?

I think of look and feel in a literal sense, as something real and tactile that the users can "look" at, then "feel"—something they can touch or interact with. Look and feel is used to evoke action—how the user will

use an interface. Messaging is holistic, as the expectation the users will have about how you will address their context. It is easy to confuse the two, because “feel” can be interpreted to mean our emotional reaction to design and the role of messaging I prefer to keep the concept of look and feel grounded in a tangible design, something I can clearly describe and show to users. I often find myself explaining the look and feel with the word “because,” with a cause-and-effect rationale for design decisions, as in “The user will press this button because...” or “The user will go to this screen because…” followed by a reason why a button or control is designed a certain way.

Establishing a look and feel usually comes from wherever design inspiration comes from. However, your personal inspiration can be a hard thing to justify. Therefore we have “design patterns,” or documented solutions to design problems, sometimes referred to as style guides. On large mobile projects or in companies with multiple designers, a style guide or pattern library is crucial, maintaining consistency in the look and feel and reducing the need for each design decision to be justified. For example, in Figure 3-16 you can see the site Pattern Tap, which is a visual collection of many user interface patterns meant for websites and web applications, but there is no reason why it can’t serve as inspiration for your mobile projects as well.

In Figure 3-17 you can see an example of a mobile design pattern. Although a lot of elements go into making Apple’s App Store successful, the most important design element is how it looks and feels. Apple includes a robust user interface tool that enables developers to use prebuilt components, supported with detailed Human Interface Guidelines (or HIG) of how to use them, similar to a pattern library.

This means that a developer can just sit down and create an iPhone application that looks like it came from Apple in a matter of minutes. During the App Store submission process, Apple then ensures that the developer uses these tools correctly according to the HIG.

The look and feel can either be consistent with the stock user interface elements that Apple provides; they can be customized, often retaining the “spirit” of Apple’s original design; or an entirely new look and feel can be defined—this approach is often used for immersive experiences.

The stock user experience that Apple provides is a great example of how look and feel works to supporting messaging. For the end user, the

design sends a clear message: by using the same visual interface metaphors that Apple uses throughout the iPhone, I can expect the action, or how this control will behave, but I can also expect the same level of quality. This invokes the message of trust and quality in the application and in the platform as a whole. Apple isn't the first to use this shared look and feel model in mobile—in fact, it is incredibly common with most smartphone platforms—but they are surely making it incredibly successful, with a massive catalog of apps and the sales to support it.

My advice to would-be mobile designers is be creative and remember the context. Like in the early days of the Web, people tend to be skeptical about mobile experiences. The modal context of the user—in this case, what device he is using—should be considered during the design, as it will help to establish the user's expectations of the experience.



Figure 3-16 Pattern Tap shows a number of user interface patterns that help to establish look and feel

You can leverage this trust to your advantage, or you can strike out on your own and forge your own metaphors. As long as you know your users and the preferred mode of context, you can create a look and feel that is right for them.

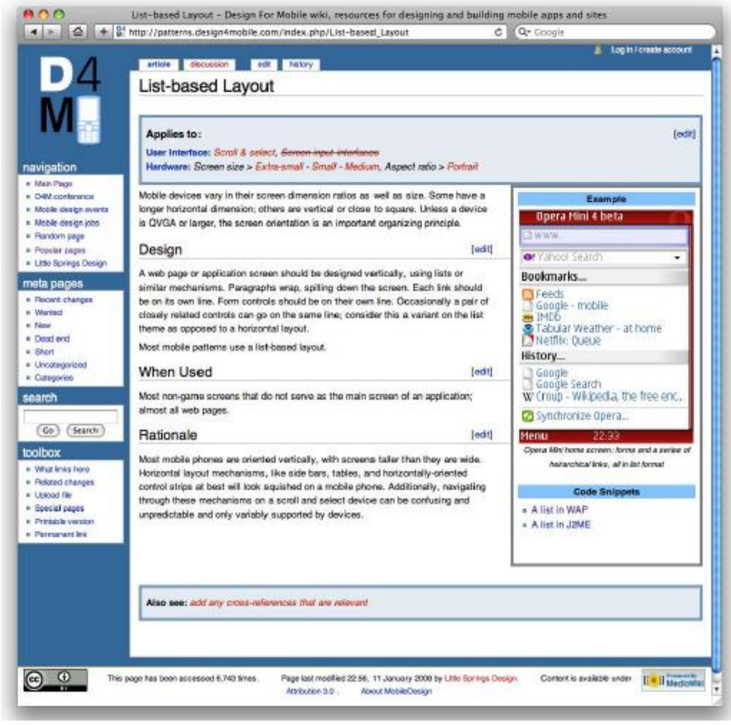


Figure 3-17 Design4Mobile provides a list of common mobile design patterns

Layout

Layout is an important design element, because it is how the user will visually process the page, but the structural and visual components of layout often get merged together, creating confusion and making your design more difficult to produce.

The first time layout should rear its head is during information architecture. In fact, I prefer to make about 90 percent of my layout decisions during the information architecture period. I ask myself questions like: where should the navigation go on the page or screen? What kind of navigation type should I use? Should I use tabs or a list? What about a sidebar for larger screens? All of these should be answered when defining the information architecture and before you begin to design. Why define the layout before the mobile design? Design is just too subjective of an issue. If you are creating a design for anyone but yourself, chances are good that there will be multiple loosely-based-on-experience opinions that will be offered and debated.

There is no right answer—only opinions and gut instincts. Plus, in corporate environments you have internal politics you have to consider,

where the design opinions of the CEO or Chief Marketing Officer (CMO) might influence a design direction more than, say, the Creative Director or Design Director.

By defining design elements like layout prior to actually applying the look and feel, you can separate the discussion. As a self-taught designer, I started out in this business making designs for my own projects. I could just put pen to paper and tweak it to my heart's content. If I wanted to radically change the layout, I could. When I started my mobile design career with my first mobile company more than a decade ago, I realized that this approach didn't work. The majority of comments that reviewers would make were about the layout. They focused on the headers, the navigation, the footer, or how content blocks are laid out, and so on. But their feedback got muddied with the "look and feel, the colors, and other design elements."

Reviewers do make remarks like "I like the navigation list, but can you make it look more raised?" Most designers don't hear that; they hear "The navigation isn't right, do it again." But, with this kind of feedback, there are two important pieces of information about different types of design. First, there is confirmation that the navigation and layout are correct. Second, there is a question about the "look and feel." Because designers hear "Do it again," they typically redo the layout, even though it was actually fine.

Creating mobile designs in an environment with multiple reviewers is all about getting the right feedback at the right time. Your job is to create a manifestation of a shared vision. Layout is one of the elements you can present early on and discuss independently. People confuse the quality and fidelity of your deliverables as design. By keeping it basic, you don't risk having reviewers confuse professionalism with design.

The irony is that as I become more adept at defining layouts, I make them of increasingly lower fidelity. For example, when I show my mobile design layouts as wireframes during the information architecture phase, I intentionally present them on blueprint paper, using handwriting fonts for my annotations (Figure 3-18). It also helps to say that this is not a design, it is a layout, so please give me feedback on the layout.

Different layouts for different devices

The second part of layout design is how to visually represent content. In mobile design, the primary content element you deal with the is navigation. Whether you are designing a site or app, you need to provide

users with methods of performing tasks, navigating to other pages, or reading and interacting with content. This can vary, depending on the devices you support.

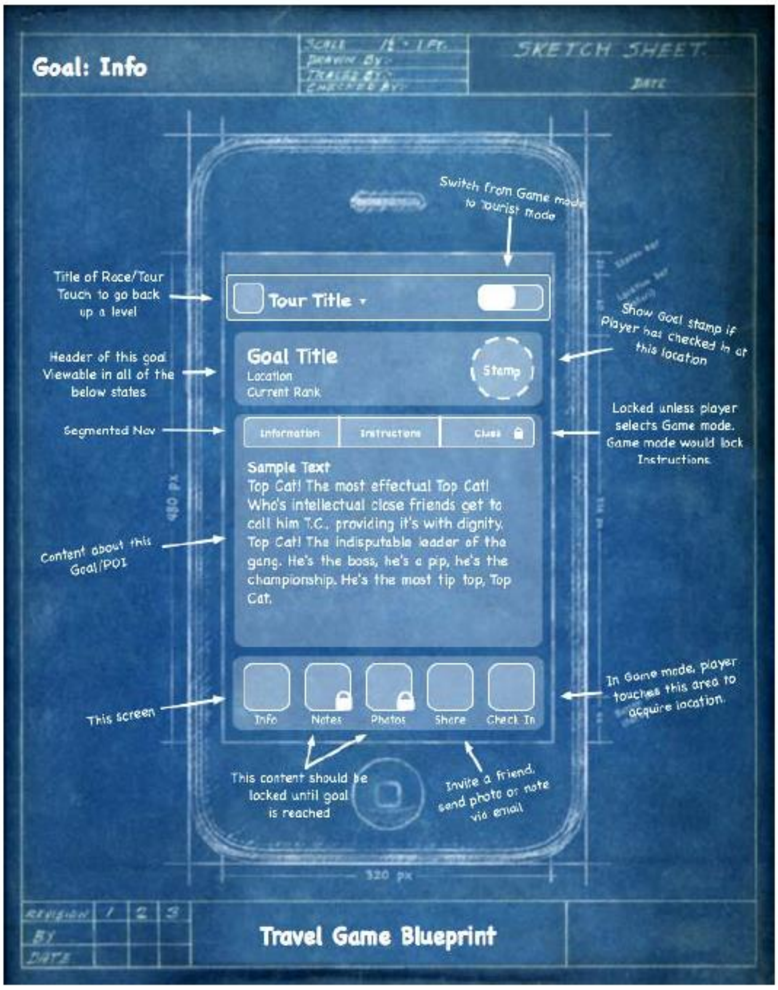


Figure 3-18 Using a low-fidelity wireframe to define the layout design element before visual design

There are two distinct types of navigation layouts for mobile devices: touch and scroll. With touch, you literally point to where you want to go; therefore, navigation can be anywhere on the screen. But we tend to see most of the primary actions or navigation areas living at the bottom of the screen and secondary actions living at the top of the screen, with the area in between serving as the content area, like what is shown in Figure 3-19.



Figure 3-19 iPhone HIG, showing the layout dimensions of Safari on the iPhone

This is the opposite of the scroll navigation type, where the device's D-pad is used to go left, right, up, or down. When designing for this type of device, the primary and often the secondary actions should live at the top of the screen. This is so the user doesn't have to press down dozens of times to get to the important stuff. In Figure 3-20, you can actually see by the bold outline that the first item selected on the screen is the link around the logo.

When dealing with scroll navigation, you also have to make the choice of whether to display navigation horizontally or vertically. Visually, horizontally makes a bit more sense, but when you consider that it forces the user to awkwardly move left and right, it can quickly become a bit cumbersome for the user to deal with. There is no right or wrong way to do it, but my advice is just to try and keep it as simple as possible.

Fixed versus fluid

Another layout consideration is how your design will scale as the device orientation changes, for example if the device is rotated from portrait mode to landscape and vice versa. This is typically described as either being fixed (a set number of pixels wide), or fluid (having the ability to scale to the full width of the screen regardless of the device orientation).



Figure 3-20 Example layout of a scroll-based application where the user had to press the D-pad past each link to scroll the page

Orientation switching has become commonplace in mobile devices, and your design should always provide the user with a means to scale the interface to take full advantage of screen real estate.

Color

The fifth design element, color, is hard to talk about in a black-and-white book. Maybe it is fitting, because it wasn't that long ago that mobile screens were available only in black and white (well, technically, it was black on a green screen). These days, we have nearly the entire spectrum of colors to choose from for mobile designs.

The most common obstacle you encounter when dealing with color is mobile screens, which come in a number of different color or bit depths, meaning the number of bits (binary digits) used to represent the color of a single pixel in a bitmapped image. When complex designs are displayed on different mobile devices, the limited color depth on one device can cause banding, or unwanted posterization in the image.

For an example of posterization, the technical term for when the gradation of tone is replaced with regions of fewer tones, see in Figure 3-

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.
21 how dramatically the color depth can affect the quality of a photo or gradient, producing banding in several parts in the image.



Figure 3-21 An example of different levels of posterization that can occur across multiple device color depths

Different devices have different color depths. In Table 3-1, you can see the supported colors and a few example devices.

Table 3-1 Supported colors and example devices

Bit depth	Supported colors	Description	Example devices
12-bit	4,096 colors	Used with older phones; dithering artifacts in photos can easily be seen.	Nokia 6800
16-bit	65,536 colors	Also known as HighColor; very common in today's mobile devices. Can cause some banding and dithering artifacts in some designs.	HTC G1, BlackBerry Bold 9000, Nokia 6620
18-bit	262,144 colors	Used in mobile devices to offer Truecolor (see following entry) levels through dithering. Limited banding may be seen.	Samsung Alias, Sony Ericsson TM506
24-bit	16.7 million colors	Also known as Truecolor; supports millions of colors and produces little banding.	iPhone, Palm Prē, Nokia N97

The psychology of color

People respond to different colors differently. It is fairly well known that different colors produce different emotions in people, but surprisingly few talk about it outside of art school. Thinking about the emotions that colors evoke in people is an important aspect of mobile design, which is such a personal medium that tends to be used in personal ways. Using the right colors can be useful for delivering the right message and setting expectations.

One of the examples I used earlier was the ESPN mobile site, which uses a bold red header to create a stark and prominent tone to the

design. But what does that say about ESPN? What does it tell the user about the experience?

For the purposes of reference, Table 3-2 provides some of the characteristics of various colors that naturally evoke certain emotions in people.

Table 3-2 Color characteristic

Color	Represents
White	Light, reverence, purity, truth, snow, peace, innocence, cleanliness, simplicity, security, humility, sterility, winter, coldness, surrender, fearfulness, lack of imagination, air, death (in Eastern cultures), life, marriage (in Western cultures), hope, bland
Black	Absence, modernity, power, sophistication, formality, elegance, wealth, mystery, style, evil, death (in Western cultures), fear, seriousness, conventionality, rebellion, anarchism, unity, sorrow, professionalism
Gray	Elegance, humility, respect, reverence, stability, subtlety, wisdom, old age, pessimism, boredom, decay, decrepitude, dullness, pollution, urban sprawl, strong emotions, balance, neutrality, mourning, formality
Yellow	Sunlight, joy, happiness, earth, optimism, intelligence, idealism, wealth (gold), summer, hope, air, liberalism, cowardice, illness (quarantine), fear, hazards, dishonesty, avarice, weakness, greed, decay or aging, femininity, gladness, sociability, friendship
Green	Intelligence, nature, spring, fertility, youth, environment, wealth, money (U.S.), good luck, vigor, generosity, go, grass, aggression, coldness, jealousy, disgrace (China), illness, greed, drug culture, corruption (North Africa), life eternal, air, earth (classical element), sincerity, renewal, natural abundance, growth
Blue	Seas, men, productiveness, interiors, skies, peace, unity, harmony, tranquility, calmness, trust, coolness, confidence, conservatism, water, ice, loyalty, dependability, cleanliness, technology, winter, depression, coldness, idealism, air, wisdom, royalty, nobility, Earth (planet), strength, steadfastness, light, friendliness, peace, truthfulness, love, liberalism (U.S. politics), and conservatism (UK, Canadian, and European politics)
Violet	Nobility, envy, sensuality, spirituality, creativity, wealth, royalty, ceremony, mystery, wisdom, enlightenment, arrogance, flamboyance, gaudiness, mourning, exaggeration, profanity, bisexuality, confusion, pride
Red	Passion, strength, energy, fire, sex, love, romance, excitement, speed, heat, arrogance, ambition, leadership, masculinity, power, danger, gaudiness, blood, war, anger, revolution, radicalism, aggression, respect, martyrs, conservatism (U.S. politics), Liberalism (Canadian politics), wealth (China), and marriage (India)
Orange	Energy, enthusiasm, balance, happiness, heat, fire, flamboyance, playfulness, aggression, arrogance, gaudiness, overemotion, warning, danger, autumn, desire
Pink	Spring, gratitude, appreciation, admiration, sympathy, socialism, femininity, health, love, romance, marriage, joy, flirtatiousness, innocence and child-like qualities
Brown	Calm, boldness, depth, nature, richness, rustic things, stability, tradition, anachronism, boorishness, dirt, dullness, heaviness, poverty, roughness, earth

Note what some of the different colors can mean in different cultures. In some cases, the color you use can have opposing meanings in different cultures. This is something to consider when thinking of deploying your mobile experience to countries with the highest number of mobile devices, such as China or India.

Color palettes

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Defining color palettes can be useful for maintaining a consistent use of color in your mobile design. Color palettes typically consist of a predefined number of colors to use throughout the design. Selecting what colors to use varies from designer to designer, each having different techniques and strategies for deciding on the colors. I've found that I use three basic ways to define a color palette:

Sequential

In this case, there are primary, secondary, and tertiary colors. Often the primary color is reserved as the "brand" color or the color that most closely resembles the brand's meaning. The secondary and tertiary colors are often complementary colors that I select using a color wheel.

Adaptive

An adaptive palette is one in which you leverage the most common colors present in a supporting graphic or image. When creating a design that is meant to look native on the device, I use an adaptive palette to make sure that my colors are consistent with the target mobile platform.

Inspired

This is a design that is created from the great pieces of design you might see online, as shown in Figure 3-22, or offline, in which a picture of the design might inspire you. This could be anything from an old poster in an alley, a business card, or some packaging. When I sit down with a new design, I thumb through some of materials to create an inspired palette. Like with the adaptive palette, you actually extract the colors from the source image, though you should never ever use the source material in a design.

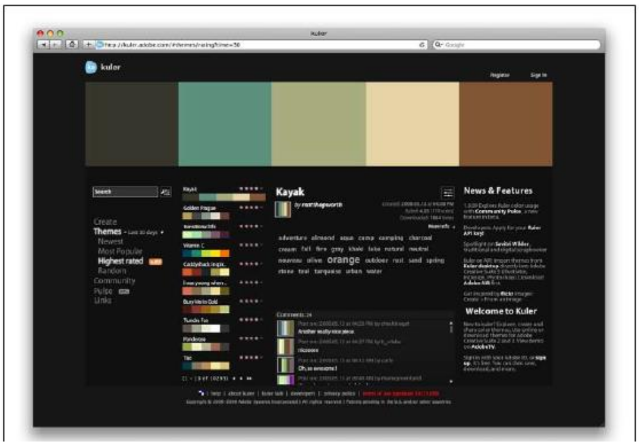


Figure 3-22 Adobe Kuler, a site that enables designers to share and use different color palattes

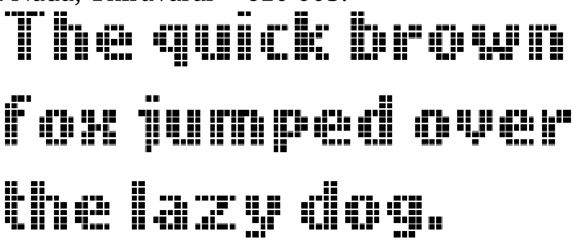


Figure 3-23 What most mobile designers think of when it comes to mobile

Typography

The sixth element of mobile design is typography, which in the past would bring to mind the famous statement by Henry Ford:
Any customer can have a car painted any color that he wants so long as it is black. Traditionally in mobile design, you had only one typeface that you could use (Figure 3-23), and that was the device font. The only control over the presentation was the size.



Figure 3-24 Microsoft Clear Type using subpixels to display sharp text

As devices improved, so did their fonts. Higher-resolution screens allowed for a more robust catalog of fonts than just the device font. First, let’s understand how mobile screens work.

Subpixels and pixel density

There seem to be two basic approaches to how type is rendered on mobile screens: using subpixel-based screens or having a greater pixel density or pixels per inch (PPI). A subpixel is the division of each pixel

into a red, green, and blue (or RGB) unit at a microscopic level, enabling a greater level of antialiasing for each font character or glyph. The addition of these RGB subpixels enables the eye to see greater variations of gray, creating sharper antialiasing and crisp text.

In Figure 3-25, you can see three examples of text rendering. The first line shows a simple black and white example, the second shows text with grayscale antialiasing, and the third line shows how text on a subpixel display would render.

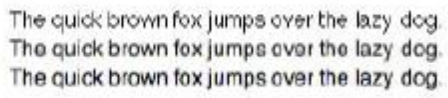


Figure 3-25 Different ways text can render on mobile screens

The Microsoft Windows Mobile platform uses the subpixel technique with its Clear- Type technology, as shown in Figure 3-26

The second approach is to use a great pixel density, or pixels per inch. We often refer to screens by either their actual physical dimensions ("I have a 15.4-inch laptop screen") or their pixel dimensions, or resolution ("The resolution of my laptop is 1440×900 pixels"). The pixel density is determined by dividing the width of the display area in pixels by the width of the display area in inches. So the pixel density for my 15.4-inch laptop would be 110 PPI. In comparison, a 1080p HD television has a PPI of 52.

As this applies to mobile devices, the higher the density of pixels, the sharper the screen appears to the naked eye. This guideline especially applies to type, meaning that as text is antialiased on a screen with a high density of tiny pixels, the glyph appears sharper to the eye. Some mobile screens have both a high PPI and subpixel technology, though these are unnecessary together. Table 3-3 provides the dimensions and PPI for a few mobile devices.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.
Table 3-3 Dimensions and PPI for some mobile devices

Mobile device	Diagonal	Pixels	PPI
Nokia N95	2.6"	240×320	153
Apple iPhone 3G	3.5"	320×480	163
Amazon Kindle	6.0"	600×800	167
HTC Dream	3.2"	320×480	181
Sony Ericsson W880i	1.8"	240×320	222
Nokia N80	2.1"	352×416	256

Type options

Fortunately, today's mobile devices have a few more options than a single typeface, but the options are still fairly limited. Coming from web design, where we have a dozen or so type options, the limited choices available in mobile design won't come as a big surprise. Essentially, you have a few variations of serif, sans-serif, and monospace fonts, and depending on the platform, maybe a few custom fonts (Figure 3-26).



Figure 3-26 Options in typography increase as the devices become more sophisticated

Therefore, when creating mobile designs for either web or native experiences, my advice is to stick with either the default device font, or

web-safe fonts—your basic serif variants like Times New Roman and Georgia or sans-serif typefaces like Helvetica, Arial, or Verdana.

Font replacement

The ability to use typefaces that are not already loaded on the device varies from model to model and your chosen platform. Some device APIs will allow you to load a typeface into your native application. Some mobile web browsers support various forms of font replacement; the two most common are sIFR and Cufon. sIFR uses Flash to replace HTML text with a Flash representation of the text, but the device of course has to support Flash. Cufon uses JavaScript and the canvas element draws the glyphs in the browser, but the device of course needs to support both JavaScript and the canvas element.

In addition, the @font-face CSS rule allows for a typeface file to be referenced and loaded into the browser, but a license for web use is usually not granted by type foundries.

Readability

The most important role of typography in mobile design is to provide the user with excellent readability, or the ability to clearly follow lines of text with the eye and not lose one's place or become disoriented, as shown in Figure 3-27. This can be done by following these six simple rules:

Use a high-contrast typeface

Remember that mobile devices are usually used outside. Having a high-contrast typeface with regard to the background will increase visibility and readability.

Use the right typeface

The type of typeface you use tells the user what to expect. For example, a sans-serif font is common in navigation or compact areas, whereas serif typefaces come in handy for lengthy or dense content areas.

Provide decent leading (rhymes with "heading") or line spacing

Mobile screens are often held 10–12" away from the eye, which can make tracking each line difficult. Increase the leading to avoid having the users lose their place.

Leave space on the right and left of each line; don't crowd the screen

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Most mobile frameworks give you full access to the screen, meaning that you normally need to provide some spacing between the right and left side of the screen's edge and your text—not much, typically about three to four character widths.

Generously utilize headings

Break the content up in the screen, using text-based headings to indicate to the user what is to come. Using different typefaces, color, and emphasis in headings can also help create a readable page.

Use short paragraphs

Like on the Web, keep paragraphs short, using no more than two to three sentences per paragraph.

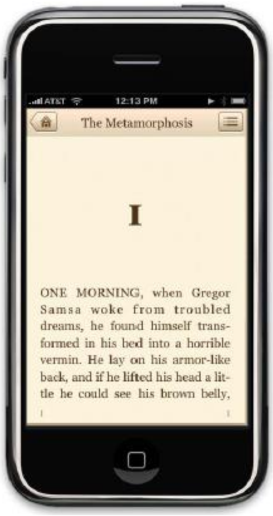


Figure 3-27 Classics, an iPhone application designed with readability and typography in mind

Graphics

The final design element is graphics, or the images that are used to establish or aid a visual experience. Graphics can be used to supplement the look and feel, or as content displayed inline with the text.

For example, in Figure 3-28, you can see Ribot's Little Spender application for the iPhone and the S60 platform. The use of graphical icons in the iPhone experience helps to establish a visual language for the user to interact with to quickly categorize entries. On the S60 application, the wallet photo in the upper-right corner helps communicate the message of the application to the user.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Iconography

The most common form of graphics used in mobile design is icons. Iconography is useful to communicate ideas and actions to users in a constrained visual space. The challenge is making sure that the meaning of the icon is clear to the user. For example, looking at Figure 3-29, you can see some helpful icons that clearly communicate an idea and some perplexing icons that leave you scratching your head.



Figure 3-28 Ribot’s Little Spender application uses graphics to define the experience

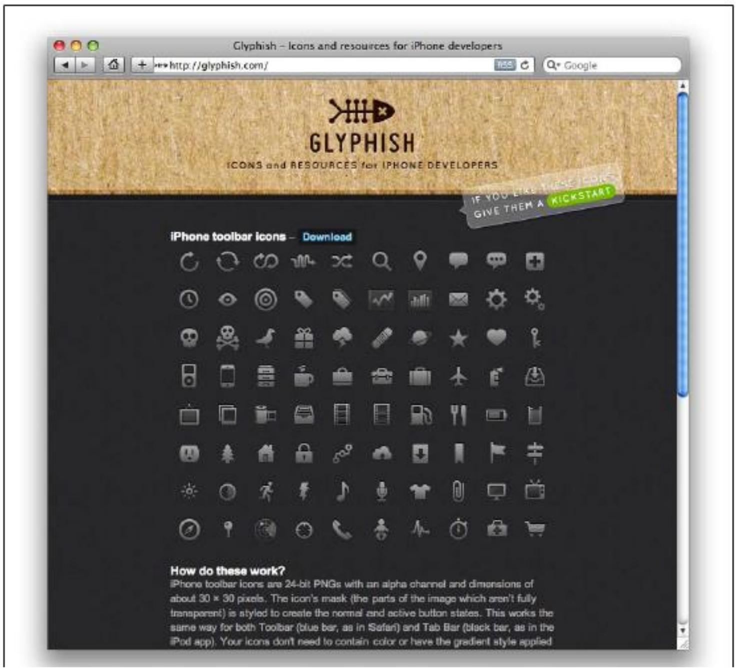


Figure 3-29 Glyphish provides free iPhone icons

Photos and images

Photos and images are used to add meaning to content, often by showing a visual display of a concept, or to add meaning to a design. Using photos and images isn't as common in mobile design as you might think. Because images have a defined height and width, they need to be scaled to the appropriate device size, either by the server, using a content adaptation model, or using the resizing properties of the device. In the latter approach, this can have a cost in performance. Loading larger images takes longer and therefore costs the user more.

Using graphics to add meaning to a design can be a useful visual, but you can encounter issues regarding how that image will display in a flexible UI—for example, when the device orientation is changed. In Figure 3-30, you can see how the pig graphic is designed to be positioned to the right regardless of the device orientation.



Figure 3-30 Using graphics in multiple device orientations

3.6 Mobile Design Tools

The closest thing to a common design tool is Adobe Photoshop, though each framework has a different method of implementing the design into the application. Some frameworks provide a complete interface toolkit, allowing designers or developers to simply piece together the interface, while others leave it to the designer to define from scratch.

In Table 3-4, you can see each of the design tools and what interface toolkits are available for it.

Table 3-4 Design tools and interface toolkits

Mobile framework	Design tool	Interface toolkits
Java ME	Photoshop, NetBeans	JavaFX, Capuchin
BREW	Photoshop, Flash	BREW UI Toolkit, uiOne, Flash
Flash Lite	Flash	Flash Lite
iPhone	Photoshop, Interface Builder	iPhone SDK
Android	Photoshop, XML-based themes	Android SDK
Palm webOS	Photoshop, HTML, CSS, and JavaScript	Mojo SDK
Mobile web	Photoshop, HTML, CSS, and JavaScript	W3C Mobile Web Best Practices
Mobile widgets	Photoshop, HTML, CSS, and JavaScript	Opera Widget SDK, Nokia Web Runtime
Mobile web apps	Photoshop, HTML, CSS, and JavaScript	iUI, jQTouch, W3C Mobile Web App Best Practices

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

3.3.1 Designing for the Right Device

With the best possible experience at hand, take a moment to relish it. Remind yourself that you are working with a rapidly evolving medium and though it might not be possible for every user to experience things exactly the way you've intended, you've set the tone and the vision for how the application should look. The truly skilled designer doesn't create just one product—she translates ideas into experiences. The spirit of your design should be able to be adapted to multiple devices.

Now is the time to ask, "What device suits this design best? What market niche would appreciate it most? What devices are the most popular within that niche?" The days of tent-poles are gone. Focus instead on getting your best possible experience to the market that will appreciate it most. It might not be the largest or best long-term market, but what you will learn from the best possible scenario will tell you volumes about your mobile product's potential for success or failure. You will learn which devices you need to design for, what users really want, and how well your design works in the mobile context.

This knowledge will help you develop your porting and/or adaptation strategy, the most expensive and riskiest part of the mobile equation.

For example, if you know that 30 percent of your users have iPhones, then that is a market you can exploit to your advantage. iPhone users consume more mobile content and products than the average mobile user. This platform has an easy-to-learn framework and excellent documentation, for both web and native products, and an excellent display and performance means. Although iPhone users might not be the majority of your market, the ability to create the best possible design and get it in front of those users presents the least expensive product to produce with the lowest risk.

With a successful single device launch, you can start to adapt designs from the best possible experience to the second best possible experience, then the third, and fourth, and so on. The best possible experience is how it should be, so it serves as a reference point for how we will adapt the experience to suit more devices.

3.6.2 Designing for Different Screen Sizes

Mobile devices come in all shapes and sizes. Choice is great for consumers, but bad for design. It can be incredibly difficult to create that best possible experience for a plethora of different screen sizes. For example, your typical feature phone might only be 140 pixels wide, whereas your higher-end smartphone might be three to four times wider.

Landscape or portrait? Fixed width or fluid? Do you use one column or two? These are common questions that come up when thinking about your design on multiple screen sizes. The bad news is that there is no simple answer. How you design each screen of content depends on the scope of devices you look to support, your content, and what type of experience you are looking to provide. The good news is that the vast majority of mobile device screens share the same vertical or portrait orientation, even though they vary greatly in dimension, as shown in Figure 3-31.

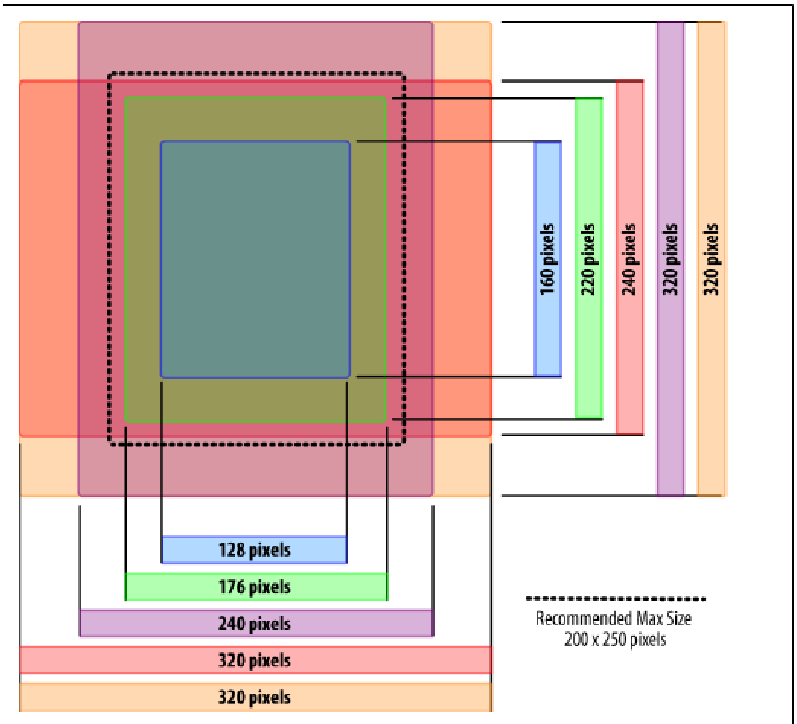


Figure 3-31 Comparing the various screen sizes

Of course, there are some devices by default in a horizontal orientation, and many smartphones that can switch between the two orientations, but most people use their mobile devices in portrait mode.

This is a big shift in thinking if you are coming from interactive design, as up to this point, screens have been getting wider, not taller.

For years now, we've become used to placing less-crucial information along the sides of web pages. In software, tasks flow from left to right. With vertical designs, the goal is to think of your design as a cascade of content from top to bottom (Figure 3-32), similar to a newspaper. The most contextual information lives at the top, and the content consumes the majority of the screen. Any exit points live at the bottom. Mobile is no different.

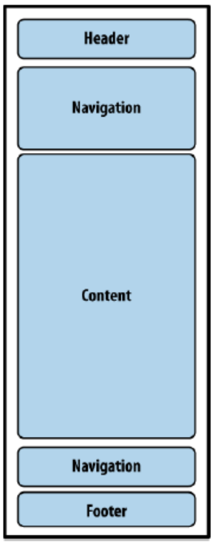


Figure 3-32The typical flow of information on mobile devices

The greatest challenge to creating a design that works well on multiple screen sizes is filling the width. For content-heavy sites and applications, the width of mobile devices is almost the perfect readability, presenting not too many words per line of text. The problem is when you have to present a number of tasks or actions. The easiest and most compatible way is to present a stacked list of links or buttons, basically one action per line. It isn't the most effective use of space, but presenting too many actions on the horizontal axis quickly clutters the design—not to mention that it is more difficult to adapt to other devices.

Unfortunately, it isn't always reasonable to implement fluid or flexible designs that stretch to fit the width of the screen. Although most mobile web browsers and device framework APIs enable it in principle, its execution across multiple devices is a little anticlimatic. Mobile websites usually employ a fixed-width layout for the lowest common denominator,

and native applications are often resized for multiple screen sizes during development.

As devices get larger, denser screens, you will see an increase in the use of touch, forcing the size of content to increase to fingertip size—typically 40 pixels wide and 40 pixels tall (Figure 3-33). This actually solves part of the horizontal axis problem, simply by making content larger for larger screens. Ironically, you can fit almost the same amount of usable content in an iPhone as you can a lower-end device.



Figure 3-33 The iPhone calculator application uses common finger-tip size cotrols

Obviously, you can fit a lot more on screen with more advanced devices, but you want to avoid forcing the user to zoom in and out of your interfaces.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

3.7 Check your Progress Questions

1. The most important MobileApp architecture decision whether to build a thin or fat mobile client
a) True
b). False

2. The ______ represent the relationship of content to other content and provide a map for how the user will travel through the informational space.

3. The ______ elements of mobile design that you need to consider, starting with the context and layering in visual elements or laying out content to achieve the design goal.

a) 5	b) 6
c) 7.	d) 9

3.8 Answers to check your progress questions.

1. a) True
2. Sitemaps
3. b) 6

3.9. Summary

The way you organize your mobile app's content is one of the main factors of its future success. If you overlook this development stage, you risk creating an app that can't provide the user experience that makes people return to the app again and again. So, choose the mobile IA patterns wisely according to your particular needs and don't be afraid to combine them. In addition, bear in mind that you need to keep your IA as simple and clear as possible, and make sure you test it and use your customer feedback properly.

3.10 Key words

Information Architecture, Prototype, Mobile Design

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

3.11 Self Assessment Questions and answers

Short Answer Questions

1. What is Information Architecture?
2. Define click streams.
3. Mention some ways of mobile prototype.
4. Write a note on color palatte.

Long Answer Questions

1. Explain about Mobile Information Architecture.
2. List and explain elements of mobile design.
3. Discuss on mobile design tools.

3.12 Further Readings

a. Brian Fling, Mobile Design and Development, OReilly media, 2009.

b.https://mobisoftinfotech.com/resources/blog/mobile-information-architecture

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

4. J2ME

4.1 Contents of the unit

4.2 Introduction

The development team at Sun worked on Java in the early 1990s to address the programming needs of the fledgling embedded computer market, but that effort was sidetracked by more compelling opportunities presented by the Internet.

As those opportunities were addressed, a new breed of portable communications devices opened other opportunities at the turn of the century. Cell phones expanded from voice communications devices to voice and text communications devices. Pocket electronic telephone directories evolved into personal digital assistants. Chipmakers were releasing new products at this time that were designed to transfer computing power from a desktop computer into mobile small computers that controlled gas pumps, cable television boxes, and an assortment of other appliances.

The time was right for the next evolution of Java. However, instead of beefing up Java with additional APIs, the team at Sun, along with the Java Community Process Program, dismantled both the Java programming language and the Java Virtual Machine. They stripped down Java APIs and the JVM to the minimum coding required providing intelligence to

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.
embedded systems and microcomputer devices. This was necessary because of resource constraints imposed upon the hardware design of these devices. The result of their efforts is J2ME.

4.3 Objectives

- To understand the overview of J2ME.
- To learn about J2ME architecture
- To learn elements needed for mobile java applications (MIDlet) development.

4.4 J2ME architecture and development environment

J2ME is a reduced version of the Java API and Java Virtual Machine that is designed to operate within the sparse resources available in the new breed of embedded computers and microcomputers.

J2ME made its debut at the JavaOne Developers Conference in mid-1999 and is targeted to developers of intelligent wireless devices and small computing devices who need to incorporate cross-platform functionality in their products.

Consumers of mobile and small computing devices have high performance expectations for these devices. They demand quick response time, compatibility with companion services, and full-featured applications in a small computing device. Consumers expect the same software and capabilities found on their desktop and laptop computers to be available on their cell phones and personal digital assistants.

To meet these expectations, developers have to rethink the way they build computer systems. Developers need to harness the power of existing front-end and back-end software found on business computers and transfer this power onto small, mobile, and wireless computing devices. J2ME enables this transformation to occur with minimal modifications, assuming that applications are scalable in design so that an application can be custom-fitted to resources available on a small computing device.

Developers seeking to build applications that run on cell phones, personal digital assistants, and various consumer and industrial appliances must strike a balance between a thick client and a thin client. A *thick client* is front-end software that contains the logic to handle a sizable amount of data processing for the system (Figure 4-1). A *thin client* is front-end

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.
software that depends on back-end software for much of the system processing (Figure 4-2).

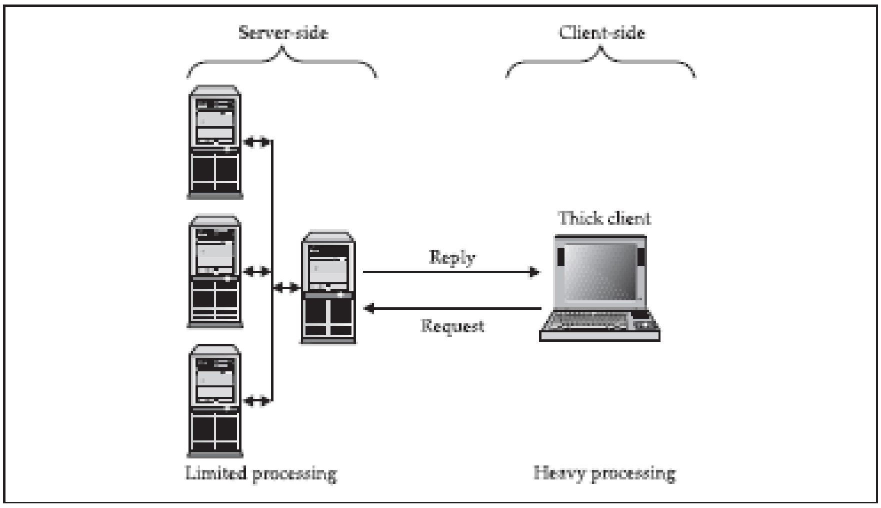


Figure 4-1. Thick client applications handle most processing locally

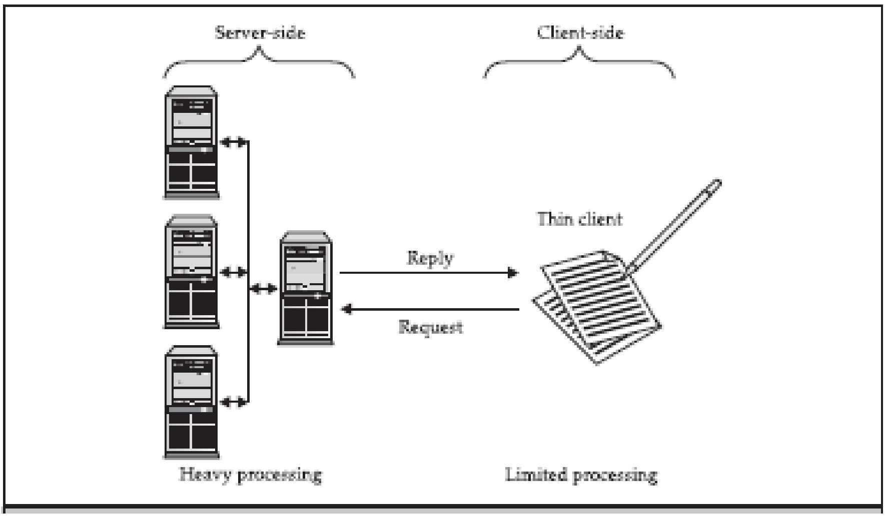


Figure 4-2. Thin client applications rely on server-side software for nearly all processing

Let's say that a wireless small computing device is used to transact orders on the floor of a stock exchange. The wireless device has software to handle user interactions such as displaying an electronic form on the screen, collecting user input, processing the input, and displaying results of the processing on the screen.

The order form is displayed on the screen, and the user enters information into the order form using various input conventions commonly found in small wireless devices. The device collects the order information and then processes the order using a combination of software on the wireless device and software running on a back-end system that receives the order through a wireless connection.

Processing on the wireless device might involve two steps: First the software performs a simple validation process to assure that all fields on the form contain information. Next the order is transmitted to the back-end system. The back-end system handles adjusting account balances and other steps involved in processing the order. A confirmation notice is returned by the back-end system to the wireless device, which displays the confirmation notice on the screen (Figure 4-3).

A key benefit of using J2ME is that J2ME is compatible with all Java-enabled devices. A Java-enabled device is any computer that runs the Java Virtual Machine.

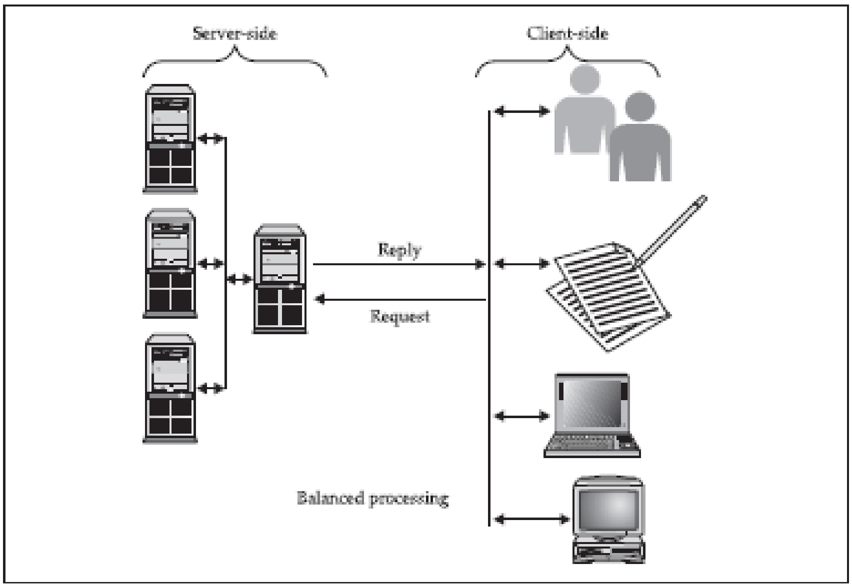


Figure 4-3 A J2ME application is a balance between local and server-side processing.

Ericson, Motorola, Nextel, Nokia, Panasonic, and RIM all have Java-enabled devices. In addition, J2ME maintains the powerful security features found in the Java language and enables wireless and small computing devices to access resources that are within an organization's firewall.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

How J2ME Is Organized

Traditional computing devices use fairly standard hardware configurations such as a display, keyboard, mouse, and large amounts of memory and permanent storage.

However, the new breed of computing devices lacks hardware configuration continuity among devices. Some devices don't have a display, permanent storage, keyboard, or mouse. And memory availability is inconsistent among small computing devices.

The lack of uniform hardware configuration among the small computing devices poses a formidable challenge for the Java Community Process Program, which is charged with developing standards for the JVM and the J2ME for small computing devices.

J2ME must service many different kinds of small computing devices, including screenphones, digital set-top boxes used for cable television, cell phones, and personal digital assistants. The challenge for the Java Community Process Program is to develop a Java standard that can be implemented on small computing devices that have nonstandard hardware configurations.

The Java Community Process Program has used a twofold approach to addressing the needs of small computing devices. First, they defined the Java run-time environment and core classes that operate on each device. This is referred to as the *configuration*. A configuration defines the Java Virtual Machine for a particular small computing device.

There are two configurations, one for handheld devices and the other for plug-in devices. Next, the Java Community Process Program defined a profile for categories of small computing devices. A *profile* consists of classes that enable developers to implement features found on a related group of small computing devices.

J2ME is targeted to developers of intelligent wireless devices and small computing devices who need to incorporate cross-platform functionality in their products. A key benefit of using J2ME is compatibility with all Java-enabled devices. Motorola, Nokia, Panasonic all have Java-enabled devices.A J2ME application is a balance between local and server-side processing. The Java Community Process Program used two approaches to addressing the needs of small computing devices.

- Configurations:It is the Java run-time environment and core classes that operate on each device. A configuration defines the Java Virtual Machine for a particular small computing device. There are two configurations.
    - CLDC for handheld devices:The CLDC (Connected Limited Device Configuration) is designed for 16-bit or 32-bit small computing devices with limited memory. These devices usually have between 160KB and 512KB of available memory. Usually these are powered by battery. They use small-bandwidth network wireless connection. These devices uses a stripped-down version of the JVM the KJava Virtual Machine (KVM). These devices include pagers, personal digital assistants, cell phones, dedicated terminals, and handheld consumer device.
    - CDC for plug-in devices:CDC(Connected Device Configuration) devices use a 32-bit architecture, have at least 2 MB of memory available, and implement a complete functional JVM. CDC devices include digital set-top boxes, home appliances, navigation systems, point-of-sale terminals, and smart phones.

- Profiles:It is defined for categories of small computing devices. A profile consists of classes that enable developers to implement features found on a related group of small computing devices. List of J2ME Profiles:
    - Profiles Used with CLDC:
        - Mobile Information Device Profile(MIDP)
        - PDA Profile(PDAP)
    - Profiles Used with CDC:
        - Foundation Profile
        - Game Profile
        - Personal Profile
        - Personal Basis Profile
        - RMI Profile.

■ The Foundation Profile is used with the CDC configuration and is the core for nearly all other profiles used with the CDC configuration because the Foundation Profile contains core Java classes.

■ The Game Profile is also used with the CDC configuration and contains the necessary classes for developing game applications for any small computing device that uses the CDC configuration.

■ The Mobile Information Device Profile (MIDP) is used with the CLDC configuration and contains classes that provide local storage, a user interface, and networking capabilities to an application that runs on a mobile computing device such as Palm OS devices. MIDP is used with wireless Java applications.

■ The PDAProfile (PDAP) is used with the CLDC configuration and contains classes that utilize sophisticated resources found on personal digital assistants. These features include better displays and larger memory than similar resources found on MIDP mobile devices (such as cell phones).

■ The Personal Profile is used with the CDC configuration and the Foundation Profile and contains classes to implement a complex user interface. The Foundation Profile provides core classes, and the Personal Profiles provide classes to implement a sophisticated user interface, which is a user interface that is capable of displaying multiple windows at a time.

■ The Personal Basis Profile is similar to the Personal Profile in that it is used with the CDC configuration and the Foundation Profile. However, the Personal Basis Profile provides classes to implement a simple user interface, which is a user interface that is capable of displaying one window at a time.

■ The RMI Profile is used with the CDC configuration and the Foundation Profile to provide Remote Method Invocation classes to the core classes contained in the Foundation Profile.

There will likely be many profiles as the proliferation of small computing devices continues. Industry groups within the Java Community Process Program (java.sun.com/aboutjava/communityprocess) define profiles. Each group establishes the standard profile used by small computing devices manufactured by that industry.

ACDC profile is defined by expanding upon core Java classes found in the Foundation Profile with classes specifically targeted to a class of small computing device. These device-specific classes are contained in a new profile that enables developers to create industrial-strength applications for those devices. However, if the Foundation Profile is specific to CDC, not all profiles are expanded upon the core classes found in the Foundation Profile.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

The modular design of the J2ME architecture enables an application to be scaled based on constraints of a small computing device. J2ME architecture doesn't replace the operating system of a small computing device. Instead, J2ME architecture consists of layers located above the native operating system, collectively referred to as the Connected Limited Device Configuration (CLDC). The CLDC, which is installed on top of the operating system, forms the run-time environment for small computing devices.

The J2ME architecture comprises three software layers (Figure 4-4). The first layer is the configuration layer that includes the Java Virtual Machine (JVM), which directly interacts with the native operating system. The configuration layer also handles interactions between the profile and the JVM. The second layer is the profile layer, which consists of the minimum set of application programming interfaces (APIs) for the small computing device. The third layer is the Mobile Information Device Profile (MIDP).

The MIDP layer contains Java APIs for user network connections, persistence storage, and the user interface. It also has access to CLDC libraries and MIDP libraries.

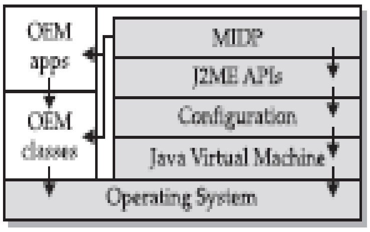


Figure 4-4. Layers of the J2ME architecture

A small computing device has two components supplied by the original equipment manufacturer (OEM). These are classes and applications. OEM classes are used by the MIDP to access device-specific features such as sending and receiving messages and accessing device-specific persistent data. OEM applications are programs provided by the OEM, such as an address book. OEM applications can be accessed by the MIDP.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

4.4.1 Small Computing Device Requirements

There are minimum resource requirements for a small computing device to run a J2ME application. First the device must have a minimum of 96 × 54 pixel display that can handle bitmapped graphics and have a way for users to input information, such as a keypad, keyboard, or touch screen. At least 128 kilobytes (KB) of nonvolatile memory is necessary to run Mobile Information Device (MID), and 8KB of nonvolatile memory is needed for storage of persistent application data. To run JVM, 32KB of volatile memory must be available. The device must also provide two-way network connectivity.

Besides minimal hardware requirements, there are also minimal requirements for the native operating system. The native operating system must implement exception handling, process interrupts, be able to run the JVM, and provide schedule capabilities.

Furthermore, all user input to the operating system must be forwarded to the JVM, otherwise the device cannot run a J2ME application. Although the native operating system doesn’t need to implement a file system to run a J2ME application, it must be able to write and read persistent data (data retained when the device is powered down) to nonvolatile memory.

4.4.2 Run-Time Environment

A MIDlet is a J2ME application designed to operate on an MIDP small computing device. A MIDlet is defined with at least a single class that is derived from the javax .microedition.midlet. MIDlet abstract class. Developers commonly bundle related MIDlets into a MIDlet suite, which is contained within the same package and implemented simultaneously on a small computing device. All MIDlets within a MIDlet suite are considered a group and must be installed and uninstalled as a group (Figure 4-5).

Members of a MIDlet suite share resources of the host environment and share the same instances of Java classes and run within the same JVM. This means if three MIDlets from the same MIDlet suite run the same class, only one instance of the class is created at a time in the Java Virtual Machine. A key benefit of the relationship among MIDlet suite members is that they share the same data, including data in persistent storage such as user preferences.

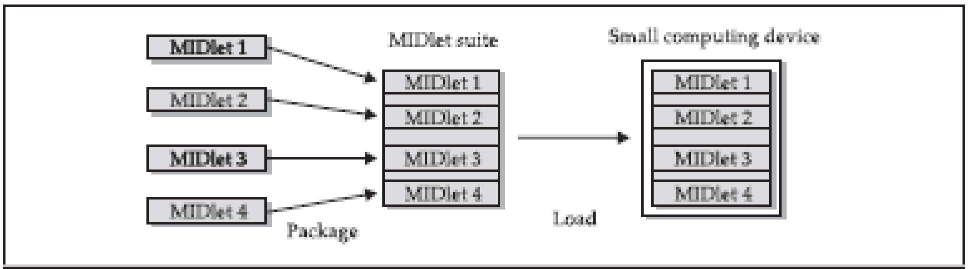


Figure 4-5. MIDlets are packaged into MIDlet suites, which are loaded in a small computing device.

This risk is reduced by synchronization primitives on the MIDlet suite level that restrict access to volatile data and persistent data. However, if a MIDlet uses multi-threading, the MIDlet is responsible for coordinated access to the record store.

Data cannot be shared between MIDlets that are not from the same MIDlet suite because the MIDlet suite name is used to identify data associated with the suite.

A MIDlet from a different MIDlet suite is considered an unreliable source. A MIDlet suite is installed, executed, and removed by the application manager running on the device. The manufacturer of the small computing device provides the application manager. Once a MIDlet suite is installed, each member of the MIDlet suite is given access to classes of the JVM and CLDC by the application manager. Likewise, a MIDlet can access classes defined in the MIDP to interact with the user interface, network, and persistent storage.

The application manager also makes the Java archive (JAR) file and the Java application descriptor (JAD) file available to members of the MIDlet suite.

Inside the Java Archive File

All the files necessary to implement a MIDlet suite must be contained within a production package called a Java archive (JAR) file. These files include MIDlet classes, graphic images (if required by a MIDlet), and the manifest file. The manifest file contains a list of attributes and related definitions that are used by the application manager to install the files contained in the JAR file onto the small computing device.

Nine attributes are defined in the manifest file; all but six of these attributes are optional. Table 4-1 lists attributes contained in a manifest file. Of these, the first six attributes are required for every manifest file. Failure to include them in the manifest file causes the application manager to halt the installation of the JAR file.

Listing 4-1 is a manifest file that contains the minimum number of attributes.

Manifest File Attribute	Description
MIDlet-Name	MIDlet suite name.
MIDlet-Version	MIDlet version number.
MIDlet-Vendor	Name of the vendor who supplied the MIDlet.
MIDlet-n	Attribute per MIDlet. Values are MIDlet name, optional icon, and MIDlet class name.
MicroEdition-Profile	Identifies the J2ME profile that is necessary to run the MIDlet.
MicroEdition-Configuration	Identifies the J2ME configuration that is necessary to run the MIDlet.
MIDlet-Icon	Icon associated with MIDlet, must be in PNG image format (optional).
MIDlet-Description	Description of MIDlet (optional).
MIDlet-Info-URL	URL containing more information about the MIDlet.

Table 4-1 Attributes of the manifest file

```
MIDlet-Name: Best MIDlet
MIDlet-Version: 2.0
MIDlet-Vendor: MyCompany
MIDlet-1: BestMIDlet, /images/BestMIDlet.png, Best.BestMIDlet
MicroEdition-Profile: MIDP-1.0
MicroEdition-Configuration: CLDC-1.0
```

Listing 4-1 Entries in manifest file

Entries in the manifest are name:value pairs and therefore can appear in any order within the manifest file. Each pair must be terminated with a carriage return. Whitespace between the colon and the attribute value is ignored when the application manager reads the manifest file.

Let’s step through the manifest file shown in Listing 4-1. The MIDlet-Name attribute specifies the name of the MIDlet suite, which is Best MIDlet in this example. The MIDlet-Version and MIDlet-Vendor attributes identify the version number of the MIDlet suite and the company or person who provided the MIDlet suite.

The MIDlet-*n* attribute contains information about each MIDlet that is in the JAR file. The number of the MIDlet replaces the letter *n*. In this example, the *n* is replaced with the digit 1 because there is only one MIDlet in the MIDlet suite.

The MIDlet-*n* attribute can contain three values that describe the MIDlet. A comma separates each value. The first value is the name of the MIDlet, which is BestMIDlet. Next is an optional value that specifies the icon that will be used with the MIDlet. In this example, BestMIDlet.png is the icon. The icon must be in the PNG image format.

And the last value for the MIDlet-*n* attribute is the MIDlet class name, which is Best.BestMIDlet. The application manager uses the class name to load the MIDlet.

The next MIDlet-*n* attribute is the MicroEdition-Profile whose value is the J2ME profile that is required to run the MIDlet. In this example the MIDP-1.0 profile is required. And the last MIDlet-*n* attribute is the MicroEdition-Configuration. The MicroEdition-Configuration attribute identifies the J2ME configuration that is necessary to run the MIDlet.

Inside the Java Application Descriptor File

You may include a Java application descriptor (JAD) file within the JAR file of a MIDlet suite as a way to pass parameters to a MIDlet without modifying the JAR file . A JAD file is also used to provide the application manager with additional content information about the JAR file to determine whether the MIDlet suite can be implemented on the device.

A JAD file is similar to a manifest in that both contain attributes that are name:value pairs. Name:value pairs can appear in any order within the JAD file. There are five required system attributes for a JAD file:

MIDlet-Name
MIDlet-Version
MIDlet-Vendor
MIDlet-*n*
MIDlet-Jar-URL

A system attribute is an attribute that is defined in the J2ME specification. Table 4-2 contains a complete list of system attributes.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.
Listing 3-2 illustrates a typical JAD file. All JAD files must have the .jad extension.

The JAD file shown in Listing 4-2 contains a few attributes that are also found in the manifest file in Listing 4-1. The first three attributes in the JAD file are identical to attributes in the manifest file.

```
MIDlet-Name: Best MIDlet
MIDlet-Version: 2.0
MIDlet-Vendor: MyCompany
MIDlet-Jar-URL: http://www.mycompany.com/bestmidlet.jar
MIDlet-1: BestMIDlet, /images/BestMIDlet.png, Best.BestMIDlet
```

Listing 4-2 A JAD file

The MIDlet-Jar-URL attribute contains the URL of the JAR file, which in this example is called bestmidlet.jar. And the last required attribute in the JAD file is the MIDlet-*n* attribute that defines a MIDlet of the MIDlet suite identical to the MIDlet-*n* attribute of the manifest. AMIDlet-*n* attribute is required for each MIDlet in the MIDlet suite.

A word of caution: the values of the MIDlet-Name, MIDlet-Version, and MIDlet- Vendor attributes in the JAD file must match the same attributes in the manifest. If the values are different, the JAR file is not installed. Other attributes that are not the same are overridden by attributes in the descriptor.

JAD File Attribute	Description
MIDlet-Name	MIDlet suite name.
MIDlet-Version	MIDlet version number.
MIDlet-Vendor	Name of the vendor who supplied the MIDlet.
MIDlet-*n*	Attribute per MIDlet. Values are MIDlet name, optional icon, and MIDlet class name.
MIDlet-Jar-URL	Location of the JAR file.
MIDlet-Jar-Size	Size of the JAR file in bytes (optional).
MIDlet-Data-Size	Minimum size (in bytes) for persistent data storage (optional).
MIDlet-Description	Description of MIDlet (optional).
MIDlet-Delete-Confirm	Confirmation required before removing the MIDlet suite (optional).
MIDlet-Install-Notify	Send installation status to given URL (optional).

Table 4-2. Attributes for a JAD file

A developer can include application attributes in a JAD file. An application attribute is a name:value pair that contains a value unique to

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.
the application. Any name can be given to an application attribute as long as it does not begin with MIDlet-.

4.4.3 MIDlet Programming

Programming a MIDlet is similar to creating a J2SE application in that you define a class and related methods. However, a MIDlet is less robust than a J2SE application because of the restrictions imposed by the small computing device. The following overview gives you a glimpse of how a MIDlet is created.

A MIDlet is a class that extends the MIDlet class and is the interface between application statements and the run-time environment, which is controlled by the application manager. A MIDlet class must contain three abstract methods that are called by the application manager to manage the life cycle of the MIDlet. These abstract methods are startApp(), pauseApp(), and destroyApp().

The startApp() method is called by the application manager when the MIDlet is started and contains statements that are executed each time the application begins execution (Figure 4-6). The pauseApp() method is called before the application manager temporarily stops the MIDlet. The application manager restarts the MIDlet by recalling the startApp() method. The destroyApp() method is called prior to the termination of the MIDlet by the application manager.

Listing 4-3 illustrates the basic shell of a MIDlet. In this example, the MIDlet class called Basic MIDlet Shell extends the MIDlet class. Any name can be used for a class as long as it conforms to the Java class naming convention.

```
public class BasicMIDletShell extends MIDlet
{
  public void startApp()
  {
  }
  public void pauseApp()
  {
  }
  public void destroyApp( boolean unconditional)
  {
  }
}
```

Listing 4-3 The basic MIDlet shell

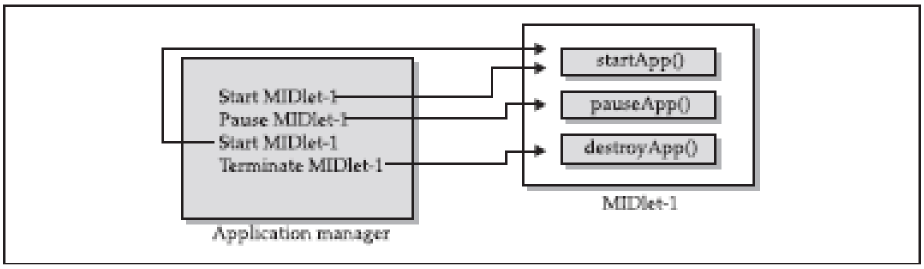


Figure 4-6. The application manager calls methods of a MIDlet

Both the startApp() and pauseApp() methods are public and have no return value nor parameter list. The destroyApp() method is also a public method without a return value. However, the destroyApp() method has a boolean parameter that is set to true if the termination of the MIDlet is unconditional, and false if the MIDlet can throw a MIDletStateChangeException telling the application manager that the MIDlet does not want to be destroyed just yet.

At the center of every MIDlet are the MIDP API classes used by the MIDlet to interact with the user and handle data management. User interactions are managed by user interface MIDP API classes. These APIs enable a developer to display screens of data and prompt the user to respond with an appropriate command. The command causes the MIDlet to execute one of three routines: perform a computation, make a network request, or display another screen.

The data-handling MIDP API classes enable the developer to perform four kinds of data routines: write and read persistent data, store data in data types, receive data from and send data to a network, and interact with the small computing device’s input/output features.

Event Handling

A MIDlet is an event-based application. All routines executed in the MIDlet are invoked in response to an event reported to the MIDlet by the application manager. The initial event that occurs is when the MIDlet is started and the application manager invokes the startApp() method.

The startApp() method in a typical MIDlet contains a statement that displays a screen of data and prompts the user to enter a selection from among one or more options. The nature and number of options is MIDlet and screen dependent.

A Command object is used to present a user with a selection of options to choose from when a screen is displayed. Each screen must have

a CommandListener. A CommandListener monitors user events with a screen and causes the appropriate code to execute based on the current event.

User Interfaces

The design of a user interface for a MIDlet depends on the restrictions of a small computing device. Some small computing devices contain resources that provide a rich user interface, while other more resource-constrained devices offer a modest user interface. A rich user interface contains the following elements, and a device with a minimal user interface has some subset of these elements as determined by the profile used for the device.

A Form is the most commonly invoked user interface element found in a MIDlet and is used to contain other user interface elements. Text is placed on a form as a StringItem, a List, a ChoiceGroup, and a Ticker.

A StringItem contains text that appears on a form that cannot be changed by the user. A List is an itemized options list from which the user can choose an option. A ChoiceGroup is a related itemized options list. And a Ticker is text that is scrollable. A user enters information into a form by using the Choice element, TextBox, TextField, or DateField elements. The Choice element returns an option that the user selected. TextBox and TextField elements collect textual information from a user and enable the user to edit information that appears in these user interface elements. The DateField is similar to a TextBox and TextField except its contents are a date and time.

An Alert is a special Form that is used to alert the user that an error has occurred. An Alert is usually limited to a StringItem user interface element that defines the nature of the error to the user.

Device Data

Small computing devices don't have the resources necessary to run an onboard database management system (DBMS). In fact some of these devices lack a file system. Therefore, a MIDlet must read and write persistent data without the advantage of a DBMS or file system.

A MIDlet can use an MIDP class—RecordStore—and two MIDP interfaces— RecordComparator and RecordFilter—to write and read persistent data. A RecordStore class contains methods used to write and

read persistent data in the form of a record. Persistent data is read from a RecordStore by using either the RecordComparator interface or the RecordFilter interface.

CDC implements the full J2SE available, but CLDC implements a stripped-down J2SE because of the limited resources in small computing devices.

Floating-point math is probably the most notable missing feature of J2ME. Floatingpoint math requires special processing hardware to perform floating-point calculations. However, most small computing devices lack such hardware and therefore are unable to process floating-point calculations. This means that your MIDlet cannot use any floating-point data types or calculations.

The second most notable difference between the Java language used in J2SE and J2ME is the absence of support for the finalize() method. The finalize() method in J2SE is automatically called before an instance of a class terminates and typically contains statements that free previously allocated resources. However, resources in a small computing device are too scarce to process the finalize() method.

Another dramatic difference is the reduced number of error-handling exceptions that are supported in J2ME. Table 4-3 lists error-handling exceptions available in J2ME. Exception handling drains system resources, which are precious in a small computing device and therefore the primary reason for trimming the number of error-handling exceptions. Typically, run-time errors are automatically responded to by the native operating system by restarting the small computing device.

Changes were also made in the Java Virtual Machine that runs on a small computing device because of resource constraints. One such change occurs with the class loader. JVM for small computing devices requires a custom class loader that is supplied by the device manufacturer and cannot be replaced or modified. Another feature lacking in the JVM is the ThreadGroup class. You cannot group threads. All threads are handled at the object level, although there is a workaround(see Chapter 4). Also, you cannot call other programming languages' methods and APIs, primarily because of the memory requirements to execute such calls. Two other features of J2SE that are missing from J2ME are weak references and the Reflection classes.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

The standard JVM uses class file verification to protect applications from malicious code through the use of a security manager. However, this process is replaced with a two-step process because of the limited resources available on small computing devices.

The first step is called pre verification and occurs outside the small computing device prior to loading the MIDlet. Preverification requires that additional attributes called stack maps are inserted into a class file by software before the second step runs. *Stack maps* describe the MIDlet's variables and operands located on the interpreter stack.

After pre verification is completed, the MIDlet class is loaded into the device, and the verifier within the small computing device validates each instruction in the MIDlet class. The MIDlet class is automatically rejected if the verifier detects an error.

	System Classes	
java.lang.Class	java.lang.Runtime	java.lang.System
java.lang.Object	java.lang.String	java.lang.Thread
java.lang.Runnable	java.lang.StringBuffer	java.lang.Throwable
	Data Type Classes	
java.lang.Boolean	java.lang.Character	java.lang.Long
java.lang.Byte	java.lang.Integer	java.lang.Short
	Collection Classes	
java.util.Enumeration	java.util.Stack	
java.util.Hashtable	java.util.Vector	
	Input/Output Classes	
java.io.ByteArrayInputStream	java.io.DataOutputStream	java.io.PrintStream
java.io.ByteArrayOutputStream	java.io.InputStream	java.io.Reader
java.io.DataInput	java.io.InputStreamReader	java.io.Writer
java.io.DataInputStream	java.io.OutputStream	
java.io.DataOutput	java.io.OutputStreamWriter	
	Calendar and Time Classes	
java.util.Calendar	java.util.Date	java.util.TimeZone

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

	Utility Classes	
java.lang.Math	java.util.Random	
	Exception Classes	
java.io.EOFException	java.lang.ClassNotFoundException	java.lang.NegativeArraySizeException
java.io.InterruptedIOException	java.lang.Exception	java.lang.NullPointerException
java.io.IOException	java.lang.IllegalAccessException	java.lang.NumberFormatException
java.io.UnsupportedEncodingException	java.lang.IllegalArgumentException	java.lang.RuntimeException
java.io.UTFDataformatException	java.lang.IllegalMonitorStateException	java.lang.SecurityException
java.lang.ArithmeticException	java.lang.IllegalThreadStateException	java.lang.StringIndexOutOfBoundsException
java.lang.ArrayIndexOutOfBoundsException	java.lang.IndexOutOfBoundsException	java.util.emptyStackException
java.lang.ArrayStoreException	java.lang.InstantiationException	java.util.NoSuchElementException
java.lang.ClassCastException	java.lang.InterruptedException	
	Error Classes	
java.lang.Error	java.lang.OutOfMemoryError	java.lang.VirtualMachineError
	Internationalization	
java.io.InputStreamReader	java.io.OutputStreamWriter	

Table 4-3 J2ME support classes

4.5 J2ME Software Development Kits

A MIDlet is built using free software packages that are downloadable from the java.sun.com web site, although you can purchase third-party development products such as Borland JBuilder Mobile Set, Sun One Studio 4 (formerly Forte for Java), and WebGain VisualCafe Enterprise Suite. Three software packages need to be downloaded from java.sun.com. These are the Java Development Kit (1.3 or greater) (java.sun.com/ j2se/downloads.html), Connected Limited Device Configuration (CLDC) (java.sun. com/products/cldc/), and the Mobile Information Device Profile (MIDP) (java.sun.com/ products/midp/).

Each of these software packages contains installation instructions that you need to follow closely in order to assure proper installation of each package. However, there are a few tips that will help you during the installation. First, install the Java development kit. The Java development kit contains the Java compiler and the jar.exe, which is used to create Java archive files as described previously in this chapter. After downloading the Java development kit package, unzip the package and run the installation program. It is best to accept the default directory, although you are free to choose a different directory for the Java development kit.

Once the Java development kit is installed, place the c:\jdk\bin directory, or whatever directory you selected for the Java development kit, on the PATH environment variable (see “Setting the Path inWindows”

sidebar). This enables you to invoke the Java compiler from anywhere on your computer.

Setting the Path in Windows

Windows 2000 and Windows NT
1. Choose System from the Control Panel.
2. Select Environment or Advanced/Environment.
3. Locate the PATH environment variable.
4. Enter the directory at the end of the path. Be sure to separate entries with a semicolon.

Windows 98 and Windows 95
1. Select Start.
2. Select Run.
3. Enter **sysedit**.
4. Select OK.
5. Locate the autoexec.bat dialog box.
6. Add the directory to the PATH environment variable.

Install the CLDC once the Java development kit is installed. Unzip the downloaded CLDC files from the java.sun.com web site onto the d:\j2me directory (J2ME_HOME) on your computer. The j2me_cldc has a bin subdirectory that contains the K Virtual Machine and the preverifier executable files for an assortment of platforms such as win32. Each platform is in its own subdirectory under j2me_cldc. Add the j2me\j2me_cldc\bin\win32 subdirectory to the PATH environment variable (see "Setting the Path in Windows" sidebar). You should substitute win32 subdirectory with the appropriate subdirectory for your platform.

Next, download and unzip the MIDP file. Be sure to use \j2me as the directory for the MIDP file. Unzipping the MIDP file creates a midp directory. The name of this directory might vary depending on the version that you download. Some versions create a midp-fcs directory, while the 1.0.3 version creates a %J2ME_HOME%\midp1.0.3fcs directory. The midp1.0.3fcs directory also contains a bin subdirectory. And you'll need to include the \j2me\midp1.0.3fcs\bin subdirectory in the PATH environment variable.

Next, create two environment variables. These are CLASSPATH and MIDP_HOME. The CLASSPATH environment variable identifies the path to be searched whenever a class is invoked. The MIDP_HOME

environment variable identifies the location of the \lib directory that contains the internal.config file and the system.config file.

Set the CLASSPATH to
d:\j2me\midp1.0.3fcs\classes;.

Notice that the CLASSPATH terminates with a period. The period implies the current directory and will cause the current directory to be searched if a class is not found in the \j2me\midp1.0.3fcs\classes directory.

Modifying the internal.config FileThe internal.config file is used to describe preferences that affect features of MIDP. Preferences are identified by name:value pairs. You can change values of name:value pairs by modifying the file with an editor. For example, MIDP contains an emulator for J2ME devices, such as cellular telephones. An emulator enables you to test the performance of your MIDlet without having to load the MIDlet into the real device. You can modify the color configuration of the emulated device by changing the value of the system.display.screen_depth attribute to 1, 2, 4, or 8. The value 1 causes the emulator to display black and white colors. The value 2 forces the emulator to display a 4-color grayscale. The value 4 displays a 15-color grayscale, and the value 8 changes the emulator to 256 possible colors.

Set the MIDP_HOME environment variable to
d:\j2me\midp1.0.3fcs

Hello World J2ME Style

You can create your first MIDlet once the Java development kit, Connected Limited

Device Configuration (CLDC), and Mobile Information Device Profile (MIDP) are installed. And keeping tradition alive, let's begin by creating a directory structure within which you can create and run MIDlets. Here are the directories that are used for examples in this chapter:

- j2me
- j2me\src
- j2me\src\greeting
- j2me\tmp_classes
- j2me\midlets

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

You'll create two MIDlets in this section, which will illustrate the basic concept of making and running a J2ME application. The first MIDlet is called HelloWorld and the other MIDlet is GoodbyeWorld. The HelloWorld MIDlet shows how to create a simple MIDlet that can be invoked directly from the class and from a Java archive file.

Later in this section you'll learn how to create a MIDlet suite that contains two MIDlets. These are HelloWorld and GoodbyeWorld.

Let's begin by creating the HelloWorld MIDlet. Enter the code shown in Listing 4-4 into a text editor such as Notepad, and save the file in the j2me\src\greeting directory as HelloWorld.java.

The HelloWorld MIDlet performs three basic functions that are found in nearly all MIDlets. These are to display a text box and a command on the screen, then listen to events that occur while the MIDlet is running.

The HelloWorld MIDlet is created by defining a class called HelloWorld that extends the MIDlet class and implements a CommandListener. The HelloWorld class contains three private data members and four methods. The data members are a Display object, a text box, and a command. The methods are startApp(), pauseApp(), and destroyApp(), which are discussed earlier in this chapter. The fourth method is called commandAction() and is invoked by the application manager whenever an event occurs.

Listing 4-4 illustrates a typical HelloWorld MIDlet. Two packages must be imported at the beginning of the MIDlet to access MIDlet classes and lcdui classes. MIDlet classes are screen oriented and create a Display object and then place components of the screen into the Display object. The Display object is then invoked later in the MIDlet to display the screen on the small computing device.

```
package greeting;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class HelloWorld extends MIDlet implements CommandListener
{
private Display display ;
private TextBox textBox ;
private Command quitCommand;
public void startApp()
```

```
{
display = Display.getDisplay(this);
quitCommand = new Command("Quit", Command.SCREEN, 1);
textBox = new TextBox("Hello World", "My first MIDlet", 40, 0);
textBox .addCommand(quitCommand);
textBox .setCommandListener(this);
display .setCurrent(textBox );
}
public void pauseApp()
{
}
public void destroyApp(boolean unconditional)
{
}
public void commandAction(Command choice, Displayable displayable)
{
if (choice == quitCommand)
{
destroyApp(false);
notifyDestroyed();
}
}
}
```

Listing 4-4 HelloWorld MIDlet source code

The Display object in this example is called display and will contain a TextBox object called textBox and a Command object called quitCommand. All three objects are private and are defined at the beginning of the HelloWorld class definition.

The startApp() method contains the necessary statements to invoke previously defined objects. The startApp() method begins by creating an instance of the Display object by calling the getDisplay() method. The instance of the Display object is assigned to the display Display object that is previously defined in the class. Calling getDisplay multiple times always returns the same Display reference for the specified MIDlet.

Next, an instance of a command object is created. There are three values required when creating a command object. The first value is the label of the command that will command, which is a screen command. The third parameter determines the priority of the command, which is the first priority—the higher the number, the lower the priority. The application

manager uses priority to determine the order in which a command appears in a menu if the MIDlet uses a menu.

The last instance of an object that is created in the startApp() is a TextBox object. Four values are necessary to create an instance of a TextBox object. The first is the caption for the TextBox object followed by the text that will appear in the TextBox object. In this example, HelloWorld is the caption and My first MIDlet is the text. The other two values are coordinates used by the application manager to position the TextBox object on the screen.

Next, the Command object must be associated with the TextBox message. This is accomplished by calling the addCommand() method of the TextBox object and passing the addCommand() method the Command object. Once the Command object is associated with the TextBox object, the CommandListener must be associated with the TextBox object in order for the CommandListener to respond to events occurring when the TextBox object is displayed on the screen. The setCommandListener() method of the TextBox object is used to associate the TextBox object with the CommandListener.

And the final statement within the startApp() method associates the TextBox object with the Display object by calling the setCurrent() method of the Display object and passing the setCurrent() method the TextBox object.

When the application manager of the small computing device runs the HelloWorld MIDlet, the startApp() method is the first method that is invoked, which causes the display that contains the Hello World message and the Quit command to be shown on the screen.

The HelloWorld MIDlet is required to define a pauseApp() method and a destroyApp() method, but these methods can remain empty because no special action is taken when the HelloWorld MIDLet is paused or destroyed.

The commandAction() method contains statements that evaluate events that occur while the HelloWorld MIDLet is running. The command selected by the user is passed to the commandAction() method as the first parameter. The second parameter is a Displayable object, which is a reference to the TextBox that is associated with the command. A TextBox along with other interface objects are Displayable objects.

An if statement is used to determine whether the user selected the Command object that is associated with the Hello World TextBox object. If so, the destroyApp() method is invoked and is passed a boolean false. The destroyApp() method is called before the MIDlet is destroyed; afterwards the notifyDestroyed() method is called to notify the application manager that the HelloWorld MIDLet has entered into the destroyed state. Prior to invoking the notifyDestroyed() method, a MIDlet should have completed its own garbage collection.

Compiling Hello World

The Hello World source code files should be saved in the new j2me\src\greeting directory as HelloWorld.java. Next, you'll need to compile the HelloWorld MIDlet.

Compiling a MIDlet is a two-step process. The first step is to use the Java compiler to transform the source file into a class file. The second step is to preverify the class file, as described previously in this chapter. The preverification generates a modified class file.

Make j2me\src\greeting the current directory, and then enter the following command at the command line. The d: drive is used in this example. You can replace the d: with the drive letter that is appropriate for your file structure.

javac -d d:\j2me\tmp_classes -target 1.1 -bootclasspath d:\j2me\midp1.0.3fcs\classes HelloWorld.java

The boot classpath option must be used when compiling a MIDlet. The boot classpath option points to the startup class files commonly referred to as the Java bootstrap files.

The startup classes are MIDP classes. If you fail to use the bootclasspath option, the compiler uses JDK classes instead of the MIDP classes. The compiler produces a file called HelloWorld.class in the j2me\tmp_classes\greeting directory. The greeting directory is created because of the package greeting declaration in the source code. The J2SDK 1.4 compiler outputs class files for JVM 1.2.

However, the pre verification expects classes for JVM 1.1. Therefore, you need to specify JVM 1.1 in the target option so the compiler generates classes for the JVM 1.1.

Next, you'll need to pre verify the HelloWorld.class that was generated by the compiler. Make sure that j2me\src\greeting is the current directory and enter the following command:

preverify -d d:\j2me\classes -classpath d:\j2me\midp1.0.3fcs\classes d:\j2me\tmp_classes

You must use two preverify options. The -d option places the class file within the tmp_classes directory. The second option is -classpath, which points to the location of the library classes that come with the MIDP. Preverification files are contained in the midp1.0.3fcs\classes directory. The output of the javac compiler is in the tmp_classes directory.

You can exclude the -classpath option if the CLASSPATH environment variable points
to the d:\j2me\midp1.0.3fcs\classes directory. In this case, you simply invoke the preverify using:
preverify -d d:\j2me\classes d:\j2me\tmp_classes

A word of caution: the preverifier overwrites the HelloWorld.class file generated by the compiler if the directory specified in the -d option is the same directory that contains the HelloWorld.class file. Replacing the HelloWorld.class file isn't a problem because the post-preverified HelloWorld.class is the file used to invoke the class.

Running Hello World

A MIDlet should be tested in an emulator before being downloaded to a small computing device. An *emulator* is software that simulates how a MIDlet will run in a small computing device. Once you're satisfied that a MIDlet is operating properly, you can deploy the MIDlet as part of a MIDlet suite.

There are two ways to run a MIDlet. These are either by invoking the MIDlet class or by creating a JAR file, then running the MIDlet from the JAR file. Let's begin by running the MIDlet class without the need of a JAR file. Make sure that j2me\src\ greeting is the current directory, and then enter the following command. Figure 4-7 illustrates how the MIDlet appears in the emulator. Click the right telephone handset icon to close the MIDlet.
midp -classpath d:\j2me\classes greeting.HelloWorld
Deploying Hello World

A MIDlet should be placed in a MIDlet suite after testing is completed. The MIDlet suite is then packaged into a JAR file along with other related files for downloading to a small computing device. This process is commonly referred to as *packaging*.

In the HelloWorld example, the MIDlet suite contains one MIDlet, which is the HelloWorld.class. Before packaging the MIDlet into a JAR file, you'll need to use an editor to create the manifest file shown in Listing 4-5. The manifest describes the JAR file. The manifest file should be saved as manifest.txt in the j2me\src\greeting directory. Notice that the MIDlet description within the manifest file contains a graphic call, /greeting/mylogo.png, that is associated with the HelloWorld MIDlet.



Figure 4-7 The HelloWorld MIDlet running in the emulator

Any PNG-formatted image file can be used in place of mylogo.png. However, all image files must be in the PNG format. You can also remove references to an image file by replacing the name of the image file with a space, such as:

MIDlet-1: HelloWorld, , greeting.HelloWorld
MIDlet-Name: Hello World
MIDlet-Version: 1.0
MIDlet-Vendor: Jim
MIDlet-1: HelloWorld, /greeting/myLogo.png, greeting.HelloWorld
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
Listing 4-5 The Manifest file for Helloworld

You can create the JAR file once the manifest.txt file is saved in the j2me\src\ greeting directory. Make sure the j2me\src\greeting directory is the current directory, and then create the JAR file by entering the following command:
jar -cfvm d:\j2me\midlets\HelloWorld.jar manifest.txt -C d:\j2me\classes greeting

The final piece of the Hello World package is a JAD file. Create the JAD file shown in Listing 4-6 using an editor, and save the JAD file in the j2me/src/greeting directory.
MIDlet-Name: Hello World
MIDlet-Version: 1.0
MIDlet-Vendor: Jim

MIDlet-Description: My First MIDlet suite
MIDlet-1: HelloWorld, /greeting/myLogo.png, greeting.HelloWorld
MIDlet-Jar-URL: HelloWorld.jar
MIDlet-Jar-Size: 1428
Listing 4-6 The JAD file for HelloWorld

Copy the HelloWorld.jad file into the j2me/midlets directory, and then make j2me/midlets the current directory. Invoke the MIDlet by entering the following command. The image of the mobile cellular telephone is displayed on the screen (Figure 4-7). Click the right telephone handset icon to close the MIDlet.
midp -classpath HelloWorld.jar -Xdescriptor HelloWorld.jad

Once you are satisfied that the MIDlet suite packaged in a JAR file is operating properly in the emulator, you can download the JAR file to a small computing device.

The downloading process is device dependent, and therefore you must refer to the device's documentation or the manufacturer's web site for steps for downloading your JAR file.

What to Do When Your MIDlet Doesn't Work Properly Sometimes a MIDlet won't compile or run properly. Although each MIDlet is unique, there are a few common problems that cause a MIDlet to fail. Here are areas to investigate if you experience a failure.

If the compiler, preverifier, JAR program, or emulator doesn't run from the command line, review the value of the PATH, CLASSPATH, and

MIDP_HOME environment variables to be sure you have included the exact path to these programs. Also make sure that the current directory reference (a period) is included in the CLASSPATH environment variable. Running out of environment space is a common problem on some platforms.

This results in not enough room to store the complete value of an environment variable such as the PATH. You can work around this problem by creating an executable file, such as a batch file in Windows, that sets the environment variables for J2ME components. Run this executable file before compiling and testing your MIDlet to temporarily reset environment variables. The environment variables return to their original values the next time you restart your computer or log in.

Many types of errors can occur during the compiling and packaging process. Some are syntax errors, which you'll be able to fix quickly by reviewing the source code. Other errors can be caused by poorly formed command line options and arguments, such as failing to insert a space between an option and a period when referencing the current directory.

Another common occurrence is for a MIDlet suite to run fine in test but fail to run after downloaded to the small computing device. In this case, the application manager on the small computing device might reject the MIDlet suite because the MIDlet suite cannot be run on the device. An oversize MIDlet suite is a likely suspect. Multiple MIDlets in a MIDlet Suite In the real world, multiple MIDlets are distributed in a single MIDlet suite. The application manager then displays each MIDlet as a menu option, enabling the user to run one of the MIDlets. Let's create another MIDlet to illustrate how to deploy a multiple MIDlet suite.

The new MIDlet is called GoodbyeWorld and is shown in Listing 4-7. Enter this code into a text editor and save the file as GoodbyeWorld.java in the j2me\src\greeting directory. Make the j2me\src\greeting directory the current directory. Compile both the HelloWorld.java and GoodbyeWorld.java files by entering the following command at the command line:

```
javac -d d:\j2me\tmp_classes -target 1.1 -bootclasspath
d:\j2me\midp1.0.3fcs\classes *.java
```

Preverify these files by entering the following command at the command line:

```
preverify -d d:\j2me\classes -classpath d:\j2me\midp1.0.3fcs\classes
```

```
d:\j2me\tmp_classes
package greeting;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class GoodbyeWorld extends MIDlet implements
CommandListener
{
private Display display ;
private TextBox textBox ;
private Command quitCommand;
public void startApp()
{
display = Display.getDisplay(this);
quitCommand = new Command("Quit", Command.SCREEN, 1);
textBox = new TextBox("Goodbye World", "My second MIDlet", 40, 0);
textBox .addCommand(quitCommand);
textBox .setCommandListener(this);
display .setCurrent(textBox );
}
public void pauseApp()
{
}
public void destroyApp(boolean unconditional)
{
}
public void commandAction(Command choice, Displayable displayable )
{
if (choice == quitCommand)
{
destroyApp(false);
notifyDestroyed();
}
}
}
```

Listing 4-7 GoodBye World Midlet Source code

Next, create a manifest.txt file, as illustrated in Listing 4-8, and save the file in the j2me/src/greeting directory. You can modify the manifest.txt file created in the previous example as an alternative to writing a new manifest file by including a description of the GoodbyeWorld class as shown in Listing 4-8.

Create the HelloWorld.jar file by entering the following command. Make sure that the j2m/src/greeting directory is the current directory. jar -cfvm d:\j2me\midlets\HelloWorld.jar manifest.txt -C d:\j2me\classes greeting You'll also be required to create or modify the existing JAD file to resemble Listing 4-9.

Save the HelloWorld.jar file in 2me/src/greeting. Next, copy the HelloWorld.jar file and the HelloWorld.jad file to the j2me/midlets directory.

Make the j2me/midlets directory the current directory, and then enter the following command on the command line to run the J2ME application:

midp -classpath HelloWorld.jar -Xdescriptor HelloWorld.jad
MIDlet-Name: Hello World
MIDlet-Version: 1.0
MIDlet-Vendor: Jim
MIDlet-1: HelloWorld, /greeting/myLogo.png, greeting.HelloWorld
MIDlet-2: GoodbyeWorld, /greeting/myLogo.png, greeting.GoodbyeWorld
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
Listing 4-8 The manifest file for HelloWorld/GoodbyeWorld MIDlet suite

MIDlet-Name: Hello World
MIDlet-Version: 1.0
MIDlet-Vendor: Jim
MIDlet-Description: My First MIDlet suite
MIDlet-1: HelloWorld, /greeting/myLogo.png, greeting.HelloWorld
MIDlet-2: GoodbyeWorld, /greeting/myLogo.png, greeting.GoodbyeWorld
MIDlet-Jar-URL: HelloWorld.jar
MIDlet-Jar-Size: 4048

Listing 4-9 The JAD file for HelloWorld/GoodbyeWorld

The cellular phone emulator displays the image of a cellular phone on the screen, as shown in Figure 4-8. Notice that the emulator's application manager displays both the HelloWorld and GoodbyeWorld MIDlets as menu options.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Click on the up or down arrow keys on the emulator to move the cursor up and down the menu options. Click on the center button to launch either the HelloWorld MIDlet or the GoodbyeWorld MIDlet. For example, if you move the cursor to the GoodbyeWorld MIDlet and select the center button on the emulator, the emulator's application manager launches the GoodbyeWorld MIDlet, as shown in Figure 4-9.

Click the left cellular telephone handset icon to return to the menu.



Figure 4-8 The HelloWorld MIDlet running in the emulator



Figure 4-9 The GoodbyeWorld MIDlet running in the emulator

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

4.6 J2ME Wireless Toolkit

Building and running a J2ME application at the command line is cumbersome, to say the least, when you are creating a robust application consisting of several MIDlets.

Creating your application within an integrated development environment is more productive than developing applications by entering commands at the command line.

There are a number of popular integrated development environments on the market designed for developing J2ME applications. These include Borland's JBuilder and Sun Microsystems' Forte. Another integrated development environment is the J2MEWireless

Toolkit that is downloadable from

java.sun.com/products/j2mewtoolkit/download.html.

The J2MEWireless Toolkit is used to develop and test J2ME applications by selecting a few buttons from a toolbar. However, the J2MEWireless Toolkit is a stripped-down integrated development environment in that it does not include an editor, a full debugger, and other amenities found in a third-party integrated development environment.

Building and Running a Project

Download the J2ME Wireless Toolkit from the Sun web site. The Toolkit file is a self extracting executable file. Run this executable after downloading the file, and the installation program creates all the directories required to run the Toolkit. The installed J2MEWireless Toolkit is placed in the WTK104 directory, although the directory might have a variation of this name depending on the version of the Toolkit that you download.

Ktoolbar is the executable within the directory that launches the Toolkit. The main window is displayed (see Figure 4-10) when you run ktoolbar. You'll notice that the main window is sparse compared with other integrated development environments.

Let's create a new project by selecting the New Project button from the toolbar. You'll be prompted to enter a project name and class name (see Figure 4-11). Enter **Hello**

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

World as the project name and **greeting.HelloWorld** as the class name, which is the name of the first MIDlet that is associated with the project.

After selecting the Create Project button, the J2ME Wireless Toolkit automatically creates a directory structure for the project and also creates the manifest file and JAD file.

You can see and modify attributes of these files by selecting the Settings option, which displays a dialog box containing a series of tabs. The first tab displayed, Required (see Figure 4-12), contains a list of attributes that are necessary for the manifest file and JAD

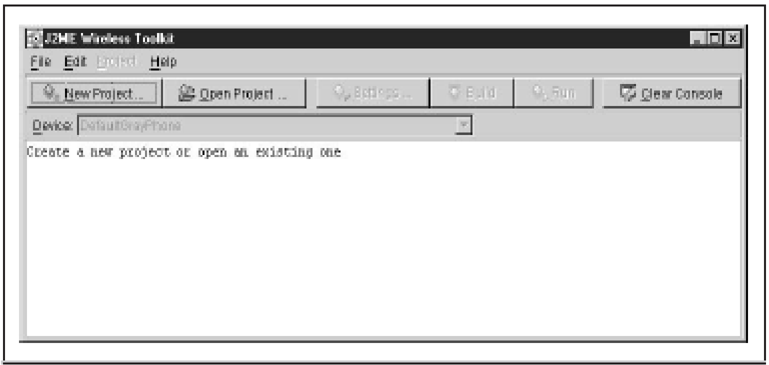


Figure 4-10. Main window of the J2ME Wireless Toolkit



Figure 4-11. Enter the project name and class name of the first MIDlet to begin the project.

The Optional tab (see Figure 4-13) contains attributes that are common to many projects but not required to build and deploy a J2ME application.

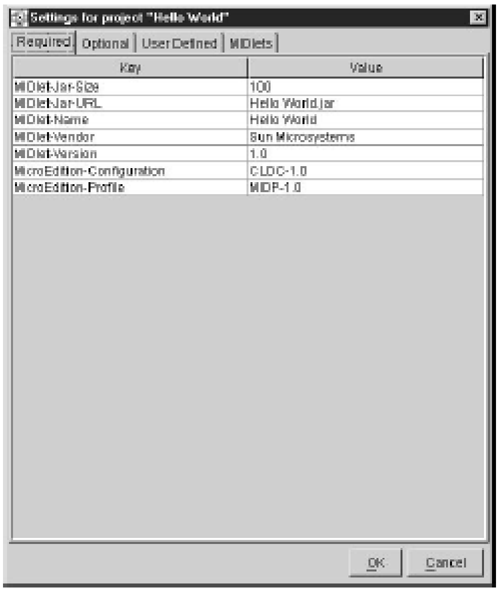


Figure 4-12. List of required attributes

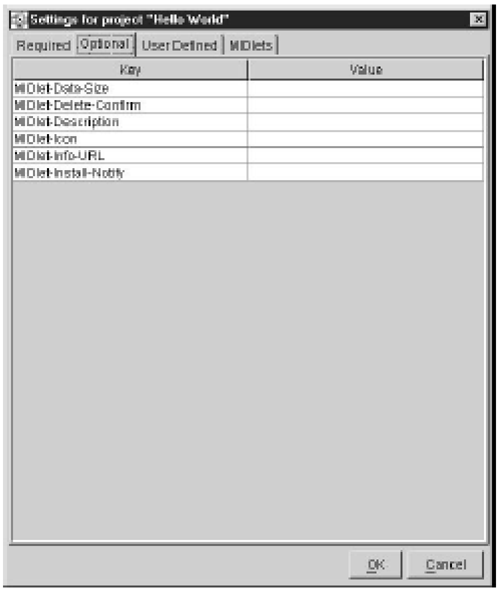


Figure 4-13 List of optional attributes

The User Defined tab (Figure 4-14) contains optional attributes specific to your application, as discussed previously in this chapter. This tab will be empty until you select the Add button and insert your own attributes. The MIDlets tab (Figure 4-15) lists MIDlets of your project.

Notice that the HelloWorld MIDlet is listed in the tab, which is the MIDlet you entered as the class name when beginning the project.

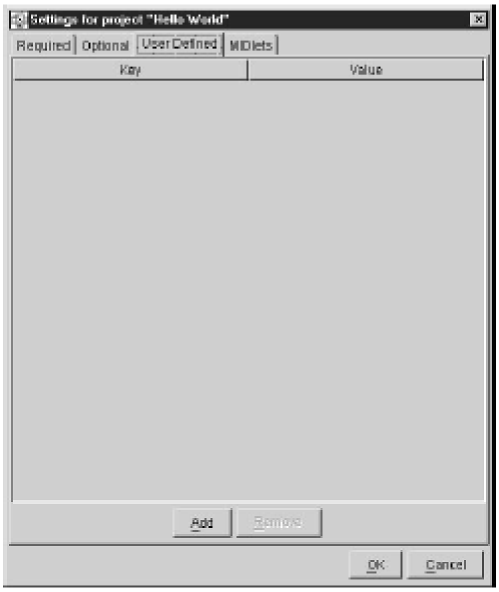
Awell-organized file structure is automatically created for your project as a result of starting a new project.Within the WTK104 directory, you'll see an apps subdirectory in which the projects you create are stored. Browse the apps subdirectory to see a subdirectory called HelloWorld, which is the name that you gave to your project. Asubdirectory of the apps directory is created for every project. And within the project's subdirectory is another set of subdirectories.

These are

- src, containing source code
- bin, containing the manifest.mf file, JAD file, and JAR file classes, containing the compiled classes
- tmpclasses, containing the preverify classes
- res, containing image, data, and other files required by the application
- Hello World Project

Let's re-create the HelloWorld and GoodbyeWorld application that you created previously in this chapter. Create a new project called Hello World following the directions in the "Hello World J2ME Style" section. Next, create a greeting directory beneath the src directory.

Copy the HelloWorld.java file and GoodbyeWorld.java file that you created previously, and place those files into the project's src\greeting subdirectory, which is Hello World\src\greeting if you named your project Hello World.



Figur 4-14. List of user-defined optional attributes

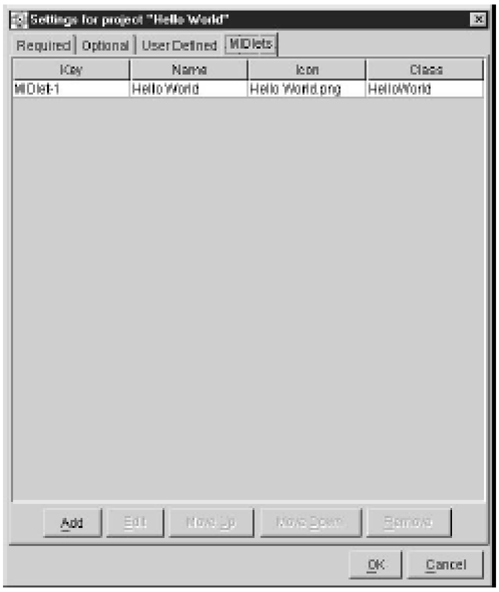


Figure 4-15.List of MIDlets that are included in the project

Select the Settings button and the MIDlets tab. You'll need to insert the GoodbyeWorld.java MIDlet into the MIDlet list. Select the Add button to display the Enter MIDlet Details dialog box (Figure 4-16), and enter **Goodbye World** as the name of the MIDlet, then **greeting. GoodbyeWorld** as the MIDlet class. Leave the Icon empty until you have

a PNG image that you want to use with the MIDlet. Select OK to return to the main screen.

If you choose not to create the greeting subdirectory, you'll need to remove the package greeting statement from the source code.

Select the Build button from the toolbar. The J2ME Wireless Toolkit compiles, preverifies, and packages the application in one step, which previously required three steps using the command line.



Figure 4-16. Enter the name of the MIDlet and the MIDlet class name

The Device drop-down box contains a list of emulators available for testing your application. Select DefaultColorPhone, and then select the Run button. The image of a color cellular telephone is displayed running your application (see Figure 4-17).



Figure 4-174 The DefaultColorPhone emulator
Rerun your application several times, alternating among device emulators. Figure 4-18 simulates your application running a DefaultGrayPhone, and Figure 4-19 emulates the Motorola i85S cellular telephone.

MIDlets on the Internet

The Wireless Toolkit can run MIDlets that access Internet resources by configuring the emulator to interact with a proxy server and let you monitor activities between the MIDlet and the Internet for debugging purposes. You configure the emulator for the Internet by selecting Edit | Preferences. The Network Configuration tab is used to set the port number and server name of the proxy server. The Trace tab is used to set preferences for monitoring the interactions between the MIDlet and the Internet. There are four options that you can set by selecting the appropriate check boxes (Figure 4-20).



Figure 4-18. The DefaultGrayPhone emulator

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.



Figure 4-19. The Motorola cellular telephone emulator

The Trace Garbage Collection option displays the status of objects that include memory allocation of existing objects, the number of objects on the heap, and the size of the largest free object. The status is displayed whenever the garbage collector is invoked.

The Trace Class Loading option will display the name of each class as it is loaded into the emulator. The Trace Class Method Calls option logs object and related methods when they are called. Display Exceptions causes all exceptions to be displayed regardless of whether they are caught or uncaught.

The Performance, Monitor, and Storage tabs are used to fine-tune the Wireless Toolkit for those aspects of an emulator.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

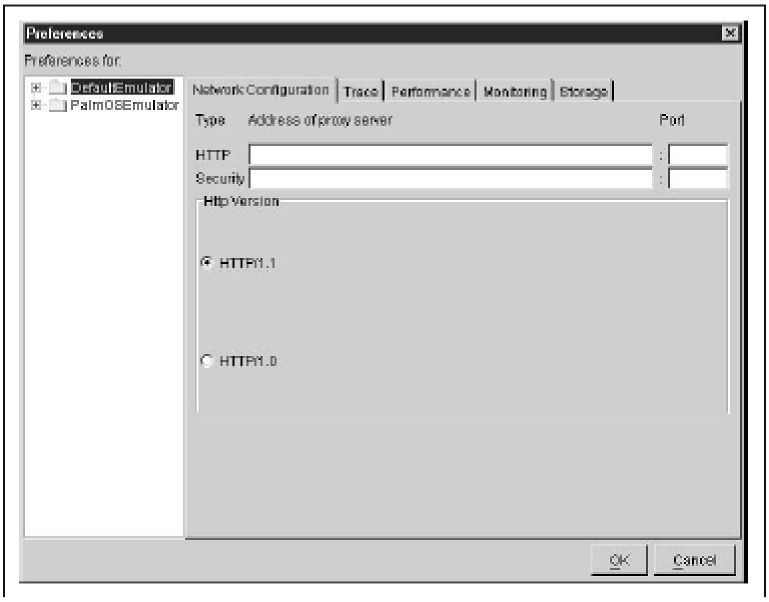


Figure 4-20. The Trace tab contains preferences for monitoring MIDlets over the Internet.

4.7. Check your progress questions

1. Choose true for
   - a. It refers as Connected, Limited Device Configuration (CLDC)
   - b. 128 kb memory for running Java, 32 kb memory for runtime memory allocation
   - c. It is used in Network connectivity, typically wireless, with low bandwidth and intermittent access
   - d. Used where we needed restricted user interface, Low power, typically battery powered
   - e. All Of Above
   - f. None

2. KVM is
   - a. Refers as K Virtual Machine is used with J2ME applications
   - b. It is used in Java application at place of JVM
   - c. Both
   - d. None

3. Choose correct order for "Generic" J2ME architecture
   - a. Profile -> Configuration-> JVM ->OS
   - b. Configuration -> Profile -> JVM ->OS

c. Profile -> JVM -> Configuration ->OS
d. None

4. Choose correct states phases MIDlet Lifecycle
   a. Active:Paused:Destroyed
   b. Paused:Active:Destroyed
   c. Paused:Destroyed:Active
   d. None

5. Which exception will thrown by destroyApp() and startApp()

   a. MIDletStateChangeException
   b. MIDletDisplayException
   c. MIDLetNotCreatedException
   d. None

4.8 Answer to check your progress Questions.

1. e
2. a
3. a
4. b
5. a

4.9 Summary

In this unit, we examined the background of J2ME and explored the J2ME configurations and profiles. We then took a look at setting up your development environment for developing J2ME applications.

We covered topics such as the K virtual machine (KVM) and the KJava profile used in conjunction with the Connected Limited Device Configuration (CDLC) API. We also discussed Mobile Information Device Profile (MIDP), which also uses CLDC. We also briefly discussed the Connected Device Configuration (CDC), which is used for larger applications.

Finally, you received hands-on experience by building a simple HelloWorld application that allowed you to see what you can do with J2ME.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

4.10. Key words

J2ME, MIDlet, Profiles Configuration, JVM, Emulator, J2ME Software Development Kits, J2ME wireless toolkit.

4.11 Self Assessment Questions and Answers

Short Answer Questions

1. What is the difference between J2ME and J2SE?
2. Define MIDlets.
3. Write the layers of J2ME architecture.
4. Outline the various challenges faced by the developer in developing applications for mobile and small computing devices.
5. Write about JVM.

Long Answer Questions

1. Draw and explain J2ME architecture.
2. Explain the J2ME architecture and the attributes of manifest file.
3. Explain in detail about 'Hello world' application using wireless toolkit.
4. What is midlet suite? Explain the MIDlet lifecycle in detail.

4.12 Further Readings

1. Topley,K. J2ME in a Nutshell -A Desktop Quick Reference
2. https://www.javaworld.com/article/2071873/mobile-java-beginning-j2me-building-midlets.html

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

5. CASE STUDY

5.1 C**ontents of the unit**

5.1 Introduction

Google acquired the Android platform in 2005 (see the sidebar "The roots of Android," later in this chapter) to ensure that a mobile operating system (OS) can be created and maintained in an open platform. Google continues to pump time and resources into the Android project. Though devices have been available only since October 2008, over a billion Android devices have now been activated, and more than a million more are being added daily. In only a few years, Android has already made a huge impact. It has never been easier for Android developers to make money by developing apps. Android users trust Google, and because your app resides in the Google Play Store, many users will be willing to trust your app, too.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

5.3 Objectives

- To Introduce the concept Google Android.
- To learn the android application development framework.
- To learn Eclipse, Emulator, Android AVD framework.
- To acquire more knowledge in Apple IOS, RIM Blackberry, Samsung Bada, Nokia Symbian, Microsoft Windows Phone.
- To learn about the development framework such as SDK, Android AVD, Emulator
- To Know about Mobile OS

5.4 Introduction to Google Android

Android is everywhere. Phones. Tablets. TVs and set-top boxes powered by Google TV. Soon, Android will be in cars and all sort of other places as well. However, the general theme of Android devices will be smaller screens and/or no hardware keyboard. And, by the numbers, Android will probably be associated mostly with smartphones for the foreseeable future. For developers, this has both benefits and drawbacks, as described next. This chapter also describes the main components in an Android application and the Android features that you can exploit when developing your applications.

5.4.1 Google Android

Android is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. Android is developed by a consortium of developers known as the Open Handset Alliance, with the main contributor and commercial marketer being Google.

Initially developed by Android Inc., which Google bought in 2005, Android was unveiled in 2007, with the first commercial Android device launched in September 2008. The current stable version is Android 10, released on September 3, 2019. The core Android source code is known as Android Open Source Project (AOSP), which is primarily licensed under the Apache License. This has allowed variants of Android to be developed on a range of other electronics, such as game consoles, digital cameras, PCs and others, each with a specialized user interface. Some well known derivatives include Android TV for televisions and Wear OS for wearables, both developed by Google.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Android's source code has been used as the basis of different ecosystems, most notably that of Google which is associated with a suite of proprietary software called Google Mobile Services (GMS), that frequently comes pre-installed on said devices. This includes core apps such as Gmail, the digital distribution platform Google Play and associated Google Play Services development platform, and usually apps such as the Google Chrome web browser. These apps are licensed by manufacturers of Android devices certified under standards imposed by Google. Other competing Android ecosystems include Amazon.com's Fire OS, or LineageOS. Software distribution is generally offered through proprietary application stores like Google Play Store or Samsung Galaxy Store, or open source platforms like Aptoide or F-Droid, which utilize software packages in the APK format.

Android has been the best-selling OS worldwide on smartphones since 2011 and on tablets since 2013. As of May 2017, it has over two billion monthly active users, the largest installed base of any operating system, and as of December 2018, the Google Play Store features over 2.6 million apps.

Android Inc. was founded in Palo Alto, California, in October 2003 by Andy Rubin, Rich Miner, Nick Sears, and Chris White. Rubin described the Android project as "tremendous potential in developing smarter mobile devices that are more aware of its owner's location and preferences". The early intentions of the company were to develop an advanced operating system for digital cameras, and this was the basis of its pitch to investors in April 2004. The company then decided that the market for cameras was not large enough for its goals, and by five months later it had diverted its efforts and was pitching Android as a handset operating system that would rival Symbian and Microsoft Windows Mobile.

Rubin had difficulty attracting investors early on, and Android was facing eviction from its office space. Steve Perlman, a close friend of Rubin, brought him $10,000 in cash in an envelope, and shortly thereafter wired an undisclosed amount as seed funding.

In July 2005, Google acquired Android Inc. for at least $50 million. Its key employees, including Rubin, Miner and White, joined Google as part of the acquisition. Not much was known about the secretive Android at the time, with the company having provided few details other than that it was making software for mobile phones. At Google, the team

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.
led by Rubin developed a mobile device platform powered by the Linux kernel. Google marketed the platform to handset makers and carriers on the promise of providing a flexible, upgradeable system. Google had "lined up a series of hardware components and software partners and signaled to carriers that it was open to various degrees of cooperation.

Speculation about Google's intention to enter the mobile communications market continued to build through December 2006. An early prototype had a close resemblance to a BlackBerry phone, with no touchscreen and a physical QWERTY keyboard, but the arrival of 2007's Apple iPhone meant that Android "had to go back to the drawing board". Google later changed its Android specification documents to state that "Touchscreens will be supported", although "the Product was designed with the presence of discrete physical buttons as an assumption, therefore a touchscreen cannot completely replace physical buttons". By 2008, both Nokia and BlackBerry announced touch-based smartphones to rival the iPhone 3G, and Android's focus eventually switched to just touchscreens. The first commercially available smartphone running Android was the HTC Dream, also known as T-Mobile G1, announced on September 23, 2008.

In 2010, Google launched its Nexus series of devices, a lineup in which Google partnered with different device manufacturers to produce new devices and introduce new Android versions. The series was described as having "played a pivotal role in Android's history by introducing new software iterations and hardware standards across the board", and became known for its "bloat-free" software with "timely updates". At its developer conference in May 2013, Google announced a special version of the Samsung Galaxy S4, where, instead of using Samsung's own Android customization, the phone ran "stock Android" and was promised to receive new system updates fast.

The device would become the start of the Google Play edition program, and was followed by other devices, including the HTC One Google Play edition, and Moto G Google Play edition. In 2015, Ars Technica wrote that "Earlier this week, the last of the Google Play edition Android phones in Google's online storefront were listed as "no longer available for sale" and that "Now they're all gone, and it looks a whole lot like the program has wrapped up".

In June 2014, Google announced Android One, a set of "hardware reference models" that would "allow [device makers] to easily create high-quality phones at low costs", designed for consumers in developing

countries. In September, Google announced the first set of Android One phones for release in India. However, Recode reported in June 2015 that the project was "a disappointment", citing "reluctant consumers and manufacturing partners" and "misfires from the search company that has never quite cracked hardware". Plans to relaunch Android One surfaced in August 2015, with Africa announced as the next location for the program a week later. A report from The Information in January 2017 stated that Google is expanding its low-cost Android One program into the United States, although The Verge notes that the company will presumably not produce the actual devices itself.

Google introduced the Pixel and Pixel XL smartphones in October 2016, marketed as being the first phones made by Google, and exclusively featured certain software features, such as the Google Assistant, before wider rollout. The Pixel phones replaced the Nexus series, with a new generation of Pixel phones launched in October 2017.

The following table shows the various android versions, code and release date.

Version	Code name	Release date
10	10	September 3, 2019
9	Pie	August 6, 2018
8.1	Oreo	December 5, 2017
8.0		August 21, 2017
7.1	Nougat	October 4, 2016
7.0		August 22, 2016
6.0	Marshmallow	October 5, 2015
5.1	Lollipop	March 9, 2015
5.0		November 3, 2014
4.4	KitKat	October 31, 2013
4.3	Jelly Bean	July 24, 2013
4.2		November 13, 2012
4.1		July 9, 2012
4.0	Ice Cream Sandwich	October 19, 2011
2.3	Gingerbread	February 9, 2011

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

5.4.2 Android Application Development

The Android SDK gives you all the tools you need to create and test Android applications. It comes in two parts: the base tools, and version-specific SDKs and related add-ons. You can find the Android developer tools on the Android Developers web site. Download the ZIP file that is appropriate for your platform and unzip it in a logical location on your machine—no specific path is required. Windows users also have the option of running a self-installing EXE file.

Install the SDKs and Add-ons

Inside the tools/ directory of your Android SDK installation from the previous step, you will see an android batch file or shell script. If you run that, you will be presented with the Android SDK and AVD Manager, shown in Figure 5-1.

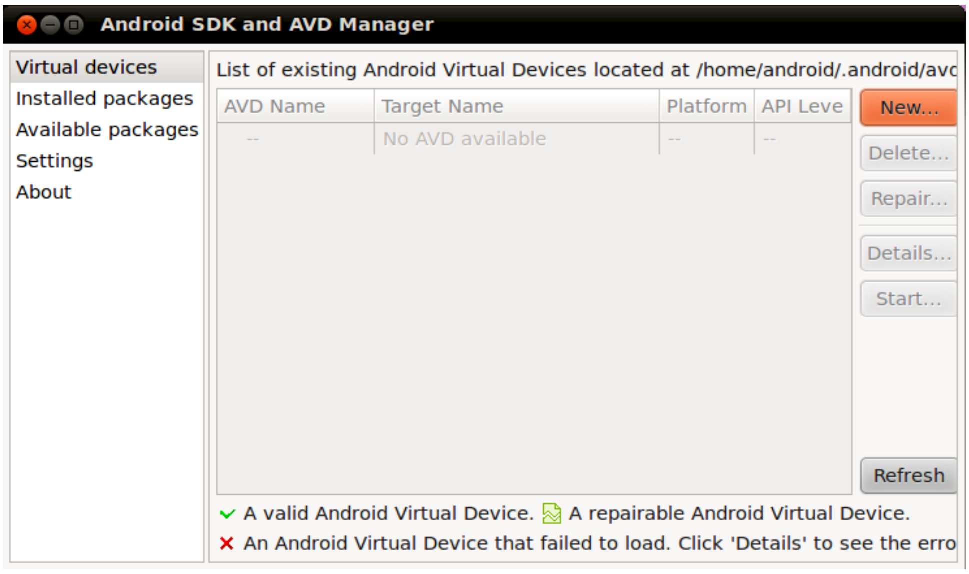


Figure 5-1. Android SDK and AVD Manager.

At this point, you have some of the build tools, but you lack the Java files necessary to compile an Android application. You also lack a few additional build tools, and the files necessary to run an Android emulator. To address this, click the Available packages option on the left to open the screen shown in Figure 5-2.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

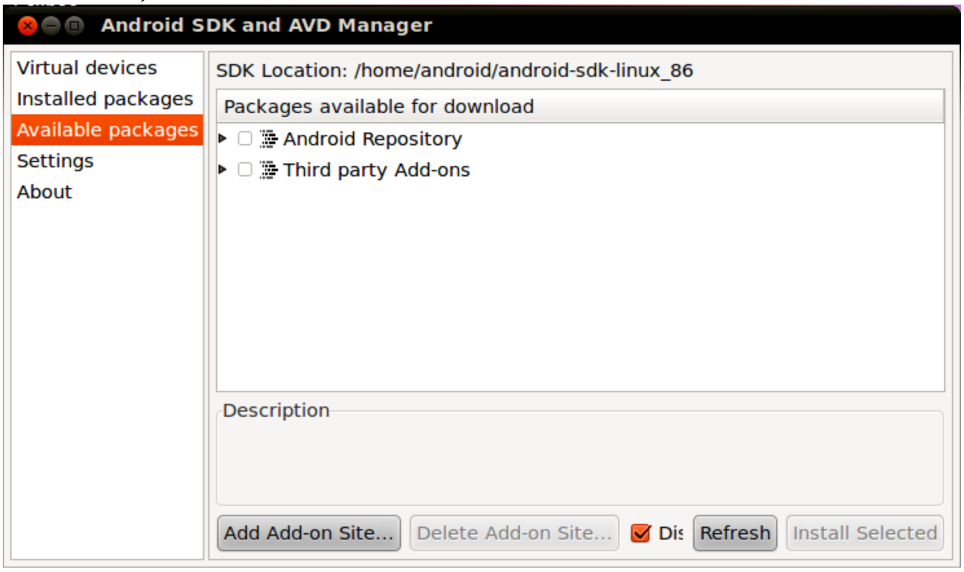


Figure 5-2. Android SDK and AVD Manager available packages

Open the Android Repository branch of the tree. After a short pause, you will see a screen similar to Figure 5-3.

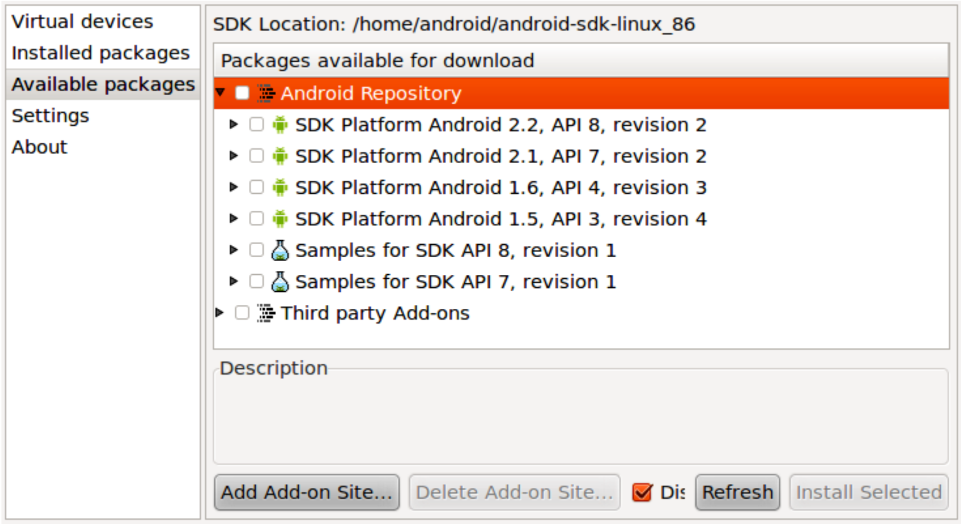


Figure 5-3. *Android SDK and AVD Manager available Android packages*

Check the boxes for the following items:

- "SDK Platform" for all Android SDK releases you want to test against
- "Documentation for Android SDK" for the latest Android SDK release
- "Samples for SDK" for the latest Android SDK release, and perhaps for older releases if you wish

Then, open the Third party Add-ons branch of the tree. After a short pause, you will see a screen similar to Figure 5.4.

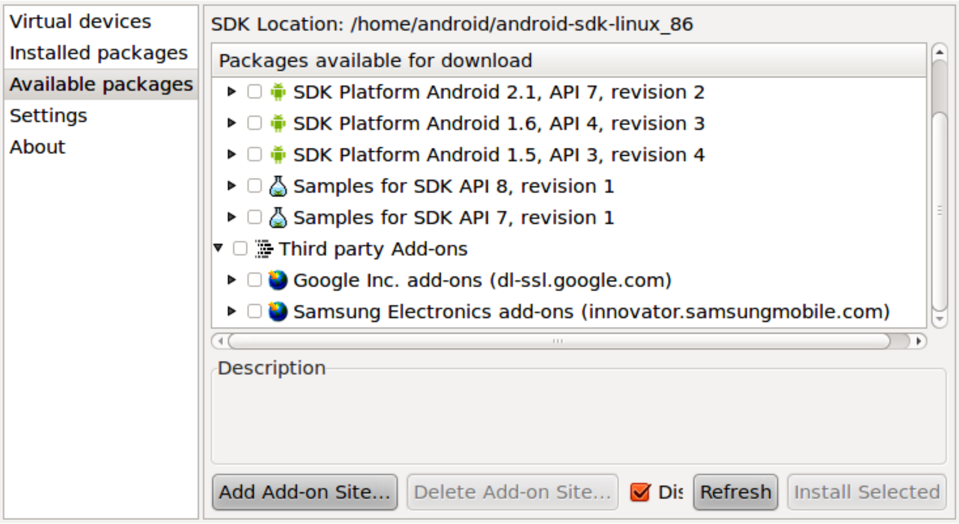


Figure 5-4. Android SDK and AVD Manager available third-party add-ons

Click the “Google Inc. add-ons” branch to open it, as shown in Figure 5-5.

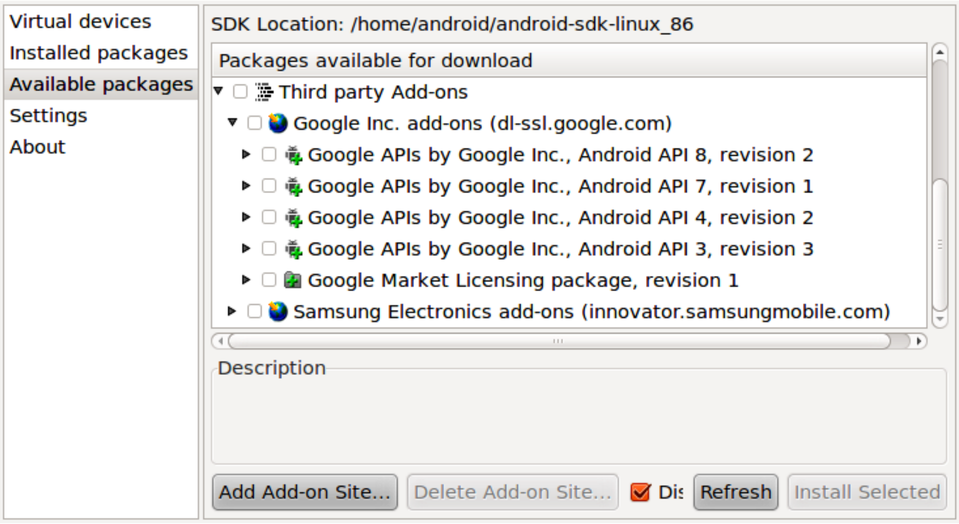


Figure 5–5. Android SDK and AVD Manager available Google add-ons

Most likely, you will want to check the boxes for the “Google APIs by Google Inc.” items that match up with the SDK versions you selected in the Android Repository branch. The Google APIs include support for Google Maps, both from your code and in the Android emulator. After you have checked all the items you want to download, click the Install Selected button, which brings up a license confirmation dialog box, shown in Figure 5-6.

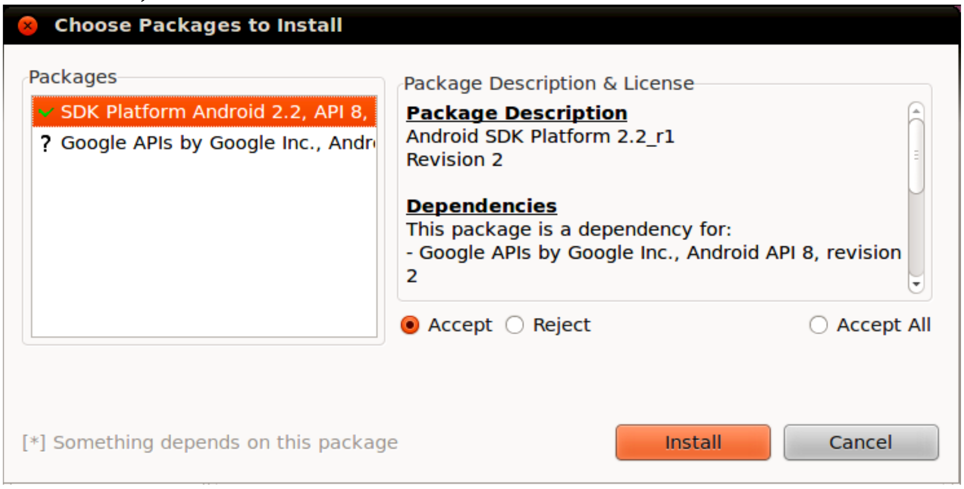


Figure 5-6. Android SDK and AVD Manger license agreement screen

Review and accept the licenses if you agree with the terms, and then click the Install button. At this point, this is a fine time to go get lunch or dinner. Unless you have a substantial Internet connection, downloading all of this data and unpacking it will take a fair bit of time.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

5.5 Development Framework

Android is a complete OS. It is not just a framework. Android is to introduce the Storage Access Framework (SAF). The SAF makes it modest for the users to browse and open documents, files and other images across all of their preferred document storage providers. Its perfect easy-to-use UI lets users browse the files and access recent in a consistent way across apps and providers. Basically this framework is coming from the view of template system. Template is like a predefined body to us.

Cloud or local storage services can participate in this ecosystem by implementing a new Documents Provider that encapsulates their own services. Client applications that need access to a provider's documents can integrate with the SAF with just a few lines of code and also some few concepts. The SAF (Storage Access Framework) includes the following process that helps us:

Document Provider:

A content provider that allows a storage service such as Google Drive to expose the records it manages. A document provider is implemented as a subclass of the Documents Provider class. The document-provider schema is based on a old-style file hierarchy though how our document provider physically stores data is up to us. The Android platform which is included to the several built-in document providers such as Downloads, Images, and Videos.

Client app:

It is a basically a traditional app that invokes the ACTION_OPEN_DOCUMENT and/or ACTION_CREATE_DOCUMENT intent. It also receives the files returned by document providers.

Picker:

The system UI that lets user's admission documents from all document providers which are content for client app's search criteria.

An Android APK is a collection of components. These components share a set of resources. Databases, preferences, files pace etc. Android component which have capability to managed lifecycle.

Some of the features offered by the SAF (Storage Access Framework) are as given below:

- Users browse the content from all document providers, not just a single application.

- Makes it possible for our application to have long term and persistent access to documents maintained by a document provider. Through this access the users can easily add, edit and delete files on the provider.
- Supports multiple user accounts and transient roots such as USB storage providers, it only appears if the USB device is plugged in.

Framework Activities and Tasks:

- The Process is started for a given user ID when is needed or require.
- It is for Binding to a Service.
- It is also used for Binding to a Content Provider
- Starting an Activity
- Firing an Intent Receiver

5.5.1SDK

Now it is the time to look at how we can start developing applications for Android OS. The Android applications are written using Java as a programming language but it is executed using a custom virtual machine that is called Dalvik rather than a traditional Java VM.

We will introduce the framework for starting with a technical explanation of the Android software stack and a look at what is included in the SDK (Software Development Kit) which is to introduction to the Android libraries and also a look at the Dalvik virtual machine system. Every individual Android application runs in the detached process within its own Dalvik instance surrendering all charge for memory and process management to the Android run time. This stops and kills processes as required to manage resources.

Dalvik and the Android run time sits on top of a Linux kernel that handles low-level hardware interaction including drivers and memory monitoring and management, while other set of APIs provides access to all of the underlying features, services and hardware.

The Android software development kit (SDK) includes everything you need to initiate development, testing and debugging of android applications.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Following are included in the SDK

- **The Android APIs**: The core of the SDK (Software Development Kit) is the Android API libraries that provide developer to access to the Android stack. These are the same libraries used at Google to create native Android applications.
- **Development Tools**: To turn Android source code into executable Android applications and the **SDK** includes several development tools that let us of the compile and debug our Android applications.
- **The Android Emulator**: The Android Emulator is a fully communicating Android device emulator featuring several alternative skins. Using the emulator that we can see how can our applications will look and behave on a real Android device (That is not a virtual concept). All Android applications run within the Dalvik Virtual Machine so that the software emulator is a brilliant situation. It is very vital point that devices are hardware-neutral. This devices are also provides a better independent test environment than any single hardware implementation.
- **Full Documentation**: The SDK (Software Development Kit) includes extensive code-level reference information detailing from exactly which we included in each package and class and also know that the using procedure them. In addition to the code documentation and Android's reference certification explicates process to get started and gives the detailed explanations of the fundamentals behind Android development System.
- **Sample Code**: The Android SDK (Software Development Kit) includes a selection of sample applications that prove some of the possibilities available using Android the process also as simple programs that highlight how to use individual API(Application Programming Interfaces) features.
- **Online Support**: Online support is no doubt very vital point that despite it's relative to youth. Android has generated a vibrant developer community. The Google Groups at groups are active forums for the Android developers with regular input from the Android development team at Google.

In that purpose we are using the popular Eclipse IDE. The Android has released a special plug-in that simplifies project creation and tightly integrates Eclipse with the Android Emulator and debugging tools.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Concept of the Android Software Stack:

The Android software stack is composed of the elements shown in Figure which is given below and we also described about it in detail below it. A Linux kernel and a collection of C/C++ libraries are exposed through an application framework that provides services to manage the run time and applications of our development area.

Linux Kernel is Core services which includes hardware drivers, process and memory management, network, security and power management which are handled by a Linux kernel. The kernel (Heart of the Linux System) also provides an abstraction layer between the hardware and the remainder of the stack.

Libraries Running on top of the kernel here Android includes various C/C++ core libraries such as libc and SSL, as well as:

- Media library for playback of audio and video media
- Surface manager to provide display management
- Graphics libraries that include SGL and OpenGL for 2D(Two Dimension) and 3D(Three Dimension) graphics.
- SQLite for native database support.
- SSL and WebKit for integrated web browser and Internet security model.

Android Run Time that makes an Android phone an Android phone rather than a mobile.
Linux implementation is the Android run time System which is including the core libraries and the Dalvik virtual machine, the Android run time is the engine that powers your applications and, along with the libraries, forms the basis for the application framework.

Core Libraries Next of them core libraries where Android development is done in Java, Dalvik is not a Java VM. The core Android libraries provide most of the functionality available in the core Java libraries as well as the Android-specific libraries.

Dalvik Virtual Machine Dalvik is a register-based virtual machine that is been optimized to ensure that a device can run multiple instances efficiently. It trusts on the Linux kernel for threading and low-level memory management.

Application Framework The application framework offers the classes used to create Android applications. It also provides a generic abstraction

for hardware access and manages the user interface and application resources.

Application Layer All applications layer both native and third party layer also are built on the application layer using the same API(Application Programing Interface) libraries. The application layer runs within the Android run time using the classes and services made available from the application framework.

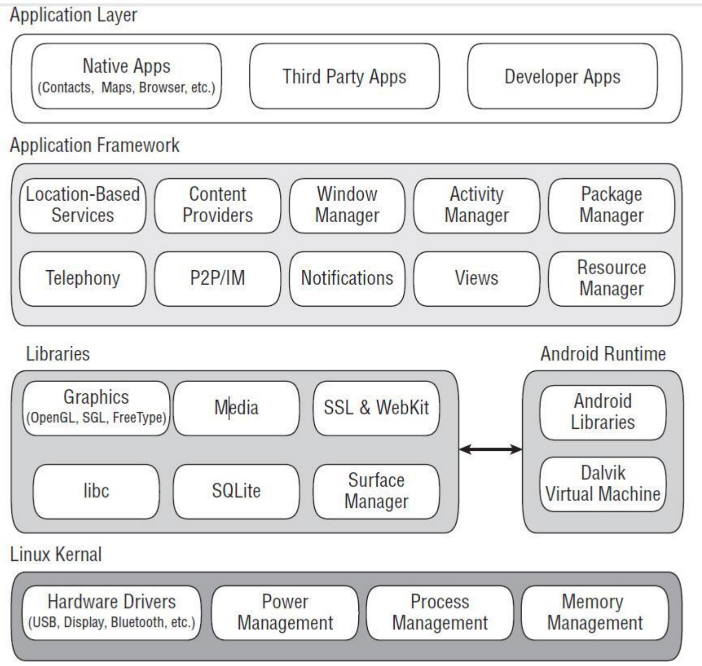


Figure 5-1: Concept of the Android Software Stack

5.5.2 Developing in Eclipse

The Android Development Tools (ADT) plugin for Eclipse adds powerful extensions to the Eclipse integrated development environment. It allows you to create and debug Android applications easier and faster. If you use Eclipse, the ADT plugin gives you an incredible boost in developing Android applications:

- It gives you access to other Android development tools from inside the Eclipse IDE. For example, ADT lets you access the many capabilities of the DDMS tool: take screenshots, manage port-forwarding, set breakpoints, and view thread and process information directly from Eclipse.

- It provides a New Project Wizard, which helps you quickly create and set up all of the basic files you'll need for a new Android application.
- It automates and simplifies the process of building your Android application.
- It provides an Android code editor that helps you write valid XML for your Android manifest and resource files.
- It will even export your project into a signed APK, which can be distributed to users.

To begin developing Android applications in the Eclipse IDE with ADT, you first need to download the Eclipse IDE and then download and install the ADT plugin. To do so, follow the steps given in Installing the ADT Plugin.

If you are already developing applications using a version of ADT earlier than 0.9, make sure to upgrade to the latest version before continuing. See the guide to Updating Your ADT Plugin.

> **Note:** This guide assumes you are using the latest version of the ADT plugin. While most of the information covered also applies to previous versions, if you are using an older version, you may want to consult this document from the set of documentation included in your SDK package (instead of the online version).

Creating an Android Project

The ADT plugin provides a New Project Wizard that you can use to quickly create a new Android project (or a project from existing code). To create a new project:

1. Select **File** > **New** > **Project**.
2. Select **Android** > **Android Project**, and click **Next**.
3. Select the contents for the project:

- Enter a *Project Name*. This will be the name of the folder where your project is created.
- Under Contents, select **Create new project in workspace**. Select your project workspace location.
- Under Target, select an Android target to be used as the project's Build Target. The Build Target specifies which Android platform you'd like your application built against.
  Unless you know that you'll be using new APIs introduced in the latest SDK, you should select a target with the lowest platform version possible.

> **Note:** You can change your the Build Target for your project at any time: Right-click the project in the Package Explorer, select **Properties**, select **Android** and then check the desired Project Target.

- Under Properties, fill in all necessary fields.
  - Enter an *Application name*. This is the human-readable title for your application — the name that will appear on the Android device.
  - Enter a *Package name*. This is the package namespace (following the same rules as for packages in the Java programming language) where all your source code will reside.
  - Select *Create Activity* (optional, of course, but common) and enter a name for your main Activity class.
  - Enter a *Min SDK Version*. This is an integer that indicates the minimum API Level required to properly run your application. Entering this here automatically sets the minSdkVersion attribute in the <uses-sdk> of your Android Manifest file. If you're unsure of the appropriate API Level to use, copy the API Level listed for the Build Target you selected in the Target tab.

2. Click **Finish**.

> **Tip:** You can also start the New Project Wizard from the *New* icon in the toolbar.

Once you complete the New Project Wizard, ADT creates the following folders and files in your new project:

src/
: Includes your stub Activity Java file. All other Java files for your application go here.

<Android Version>/ (e.g., Android 1.1/)
: Includes the android.jar file that your application will build against. This is determined by the build target that you have chosen in the *New Project Wizard*.

gen/
: This contains the Java files generated by ADT, such as your R.java file and interfaces created from AIDL files.

assets/
: This is empty. You can use it to store raw asset files.

res/
: A folder for your application resources, such as drawable files, layout files, string values, etc. See Application Resources.

AndroidManifest.xml

The Android Manifest for your project. See The AndroidManifest.xml File.

default.properties

This file contains project settings, such as the build target. This files is integral to the project, as such, it should be maintained in a Source Revision Control system. It should never be edited manually — to edit project properties, right-click the project folder and select "Properties".

5.5.3 Emulator

- An emulator acts as a real Android device (in most cases) and allows us to run and test the application without having a real device.
- The ADT plugin includes an emulator to deploy and run an Android application.

Use the Emulator to Test Different Configurations

Create multiple AVDs that each define a different device configuration with which your application is compatible, then launch each AVD into a new emulator from the SDK and AVD Manager. Set the target mode in your app's run configuration to manual, so that when you run your application, you can select from the available virtual devices.

Running your application from Eclipse will usually require just a couple clicks, whether you're running it on the emulator or on an attached device. The information below describes how to get set up and run your application from Eclipse.

Running on the emulator

Before you can run your application on the Android Emulator, you **must** create an AVD.

To run (or debug) your application, select **Run** > **Run** (or **Run** > **Debug**) from the Eclipse menu bar. The ADT plugin will automatically create a default launch configuration for the project. Eclipse will then perform the following:

1. Compile the project (if there have been changes since the last build).
2. Create a default launch configuration (if one does not already exist for the project).
3. Install and start the application on an emulator (or device), based on the Deployment Target defined by the run configuration.

   By default, Android run configurations use an "automatic target" mode for selecting a device target. For information on how automatic target mode selects a deployment target, see Automatic and manual target modes below.

If debugging, the application will start in the "Waiting For Debugger" mode. Once the debugger is attached, Eclipse will open the Debug perspective.

To set or change the launch configuration used for your project, use the launch configuration manager. See Creating a Run Configuration for information.

Be certain to create multiple AVDs upon which to test your application. You should have one AVD for each platform and screen type with which your application is compatible. For instance, if your application compiles against the Android 1.5 (API Level 3) platform, you should create an AVD for each platform equal to and greater than 1.5 and an AVD for each screen type you support, then test your application on each one.

Running on a device

Before you can run your application on a device, you must perform some basic setup for your device:

1. Declare your application as debuggable in your manifest
2. Enable USB Debugging on your device
3. Ensure that your development computer can detect your device when connected via USB

Read Setting up a Device for Development for more information.

Once set up and your device is connected via USB, install your application on the device by selecting **Run** > **Run** (or **Run** > **Debug**) from the Eclipse menu bar.

Creating a Run Configuration

The run configuration specifies the project to run, the Activity to start, the emulator or connected device to use, and so on. When you first run a project as an *Android Application*, ADT will automatically create a run configuration. The default run configuration will launch the default project Activity and use automatic target mode for device selection (with no preferred AVD). If the default settings don't suit your project, you can customize the launch configuration or even create a new.

To create or modify a launch configuration, follow these steps as appropriate for your Eclipse version:

1. Open the run configuration manager.

- In Eclipse 3.3 (Europa), select **Run** > **Open Run Dialog** (or **Open Debug Dialog**)
- In Eclipse 3.4 (Ganymede), select **Run** > **Run Configurations** (or **Debug Configurations**)

2. Expand the **Android Application** item and create a new configuration or open an existing one.

To create a new configuration:

1. Select **Android Application** and click the *New launch configuration* icon above the list (or, right-click **Android Application** and click **New**).
2. Enter a Name for your configuration.
3. In the Android tab, browse and select the project you'd like to run with the configuration.

○

To open an existing configuration, select the configuration name from the list nested below **Android Application**.

3. Adjust your desired launch configuration settings.

   In the Target tab, consider whether you'd like to use Manual or Automatic mode when selecting an AVD to run your application. See the following section on Automatic and manual target modes).
   You can specify any emulator options to the Additional Emulator Command Line Options field. For example, you could add -scale 96dpi to scale the AVD's screen to an accurate size, based on the dpi of your computer monitor. For a full list of emulator options, see the Android Emulator document.

   Automatic and manual target modes

   By default, a run configuration uses the **automatic** target mode in order to select an AVD. In this mode, ADT will select an AVD for the application in the following manner:

   1. If there's a device or emulator already running and its AVD configuration meets the requirements of the application's build target, the application is installed and run upon it.
   2. If there's more than one device or emulator running, each of which meets the requirements of the build target, a "device chooser" is shown to let you select which device to use.
   3. If there are no devices or emulators running that meet the requirements of the build target, ADT looks at the available AVDs.

If one meets the requirements of the build target, the AVD is used to launch a new emulator, upon which the application is installed and run.

4. If all else fails, the application will not be run and you will see a console error warning you that there is no existing AVD that meets the build target requirements.
5. However, if a "preferred AVD" is selected in the run configuration, then the application will *always* be deployed to that AVD. If it's not already running, then a new emulator will be launched.
6. If your run configuration uses **manual** mode, then the "device chooser" is presented every time that your application is run, so that you can select which AVD to use.

Signing your Applications

As you begin developing Android applications, understand that all Android applications must be digitally signed before the system will install them on an emulator or an actual device. There are two ways to do this: with a debug key (for immediate testing on an emulator or development device) or with a private key (for application distribution).

The ADT plugin helps you get started quickly by signing your .apk files with a debug key, prior to installing them on an emulator or development device. This means that you can quickly run your application from Eclipse without having to generate your own private key. No specific action on your part is needed, provided ADT has access to Keytool.However, please note that if you intend to publish your application, you **must** sign the application with your own private key, rather than the debug key generated by the SDK tools.

Please read Signing Your Applications, which provides a thorough guide to application signing on Android and what it means to you as an Android application developer. The document also includes a guide to exporting and signing your application with the ADT's Export Wizard.

Working with Library Projects

Library project example code

The SDK includes an example application called TicTacToeMain that shows how a dependent application can use code and resources from an Android Library project. The TicTacToeMain application uses code and resources from an example library project called TicTacToeLib.

To download the sample applications and run them as projects in your environment, use the *Android SDK and AVD Manager* to download the "Samples for SDK API 8" component into your SDK.
For more information and to browse the code of the samples, see the TicTacToeMain application.

An Android *library project* is a development project that holds shared Android source code and resources. Other Android application projects can reference the library project and, at build time, include its compiled sources in their .apk files. Multiple application projects can reference the same library project and any single application project can reference multiple library projects.

If you have source code and resources that are common to multiple application projects, you can move them to a library project so that it is easier to maintain across applications and versions.
Here are some common scenarios in which you could make use of library projects:

- If you are developing multiple related applications that use some of the same components, you could move the redundant components out of their respective application projects and create a single, reuseable set of the same components in a library project.
- If you are creating an application that exists in both free and paid versions, you could move the part of the application that is common to both versions into a library project. The two dependent projects, with their different package names, will reference the library project and provide only the difference between the two application versions.

Structurally, a library project is similar to a standard Android application project. For example, it includes a manifest file at the project root, as well as src/, res/ and similar directories. The project can contain the same types of source code and resources as a standard Android project, stored in the same way. For example, source code in the library project can access its own resources through its R class.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

However, a library project differs from an standard Android application project in that you cannot compile it directly to its own .apk or run it on the Android platform. Similarly, you cannot export the library project to a self-contained JAR file, as you would do for a true library. Instead, you must compile the library indirectly, by referencing the library from a dependent application's build path, then building that application.

When you build an application that depends on a library project, the SDK tools compile the library and merge its sources with those in the main project, then use the result to generate the .apk. In cases where a resource ID is defined in both the application and the library, the tools ensure that the resource declared in the application gets priority and that the resource in the library project is not compiled into the application .apk. This gives your application the flexibility to either use or redefine any resource behaviors or values that are defined in any library.

To organize your code further, your application can add references to multiple library projects, then specify the relative priority of the resources in each library. This lets you build up the resources actually used in your application in a cumulative manner. When two libraries referenced from an application define the same resource ID, the tools select the resource from the library with higher priority and discard the other.

ADT lets you add references to library projects and set their relative priority from the application project's Properties. As shown in Figure 2, below, once you've added a reference to a library project, you can use the **Up** and **Down** controls to change the ordering, with the library listed at the top getting the higher priority. At build time, the libraries are merged with the application one at a time, starting from the lowest priority to the highest.

Note that a library project cannot itself reference another library project and that, at build time, library projects are *not* merged with each other before being merged with the application. However, note that a library can import an external library (JAR) in the normal way.

The sections below describe how to use ADT to set up and manage library your projects. Once you've set up your library projects and moved code into them, you can import library classes and resources to your application in the normal way.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Development requirements

Android library projects are a build-time construct, so you can use them to build a final application .apk that targets any API level and is compiled against any version of the Android library.

However, to use library projects, you need to update your development environment to use the latest tools and platforms, since older releases of the tools and platforms do not support building with library projects. Specifically, you need to download and install the versions listed below:

Table 1. Minimum versions of SDK tools and plaforms on which you can develop library projects.

Component	Minimum Version
SDK Tools	r6 (or higher)
Android 2.2 platform	r1 (or higher)
Android 2.1 platform	r2 (or higher)
Android 2.0.1 platform	*not supported*
Android 2.0 platform	*not supported*
Android 1.6 platform	r3 (or higher)
Android 1.5 platform	r4 (or higher)
ADT Plugin	0.9.7 (or higher)

You can download the tools and platforms using the *Android SDK and AVD Manager*, as described in Adding SDK Components. To install or update ADT, use the Eclipse Updater as described in ADT Plugin for Eclipse.

Setting up a library project

A library project is a standard Android project, so you can create a new one in the same way as you would a new application project. Specifically, you can use the New Project Wizard, as described in Creating an Android Project, above.

When you are creating the library project, you can select any application name, package, and set other fields as needed, as shown in the diagram below. Click Finish to create the project in the workspace.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Next, set the project's Properties to indicate that it is a library project:

1. In the **Package Explorer**, right-click the library project and select **Properties**.
2. In the **Properties** window, select the "Android" properties group at left and locate the **Library** properties at right.
3. Select the "is Library" checkbox and click **Apply**.
4. Click **OK** to close the **Properties** window.
5. The new project is now marked as a library project. You can begin moving source code and resources into it, as described in the sections below.
6. You can also convert an existing application project into a library. To do so, simply open the Properties for the project and select the "is Library" checkbox. Other application projects can now reference the existing project as a library project.

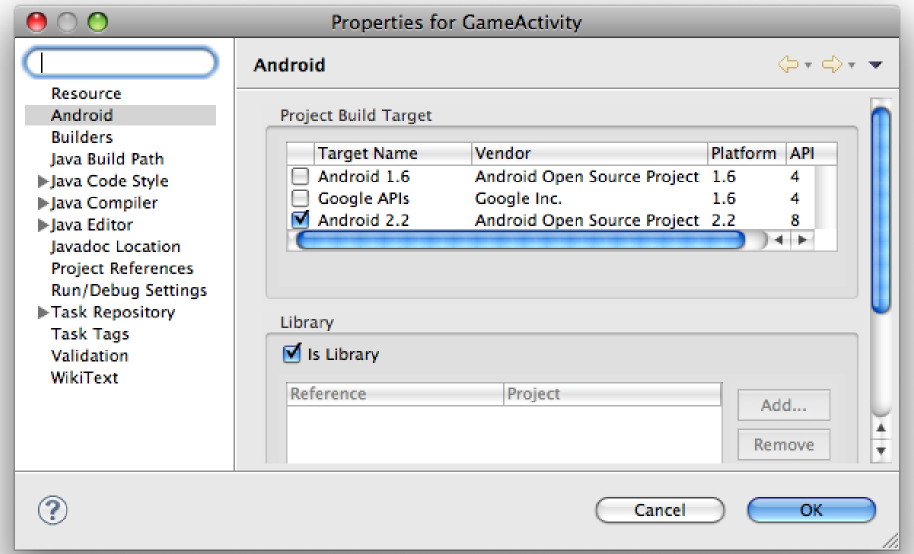


Figure 5-2. Marking a project as an Android library project.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Creating the manifest file

A library project's manifest file must declare all of the shared components that it includes, just as would a standard Android application. For example, the TicTacToeLib example library project declares the Activity GameActivity:

```
<manifest>
  ...
  <application>
    ...
    <activity android:name="GameActivity" />
    ...
  </application>
</manifest>
```

Referencing a library project from an application

If you are developing an application and want to include the shared code or resources from a library project, you can do so easily by adding a reference to the library project in the application project's Properties.

To add a reference to a library project, follow these steps:

1. In the **Package Explorer**, right-click the dependent project and select **Properties**.
2. In the **Properties** window, select the "Android" properties group at left and locate the **Library** properties at right.
3. Click **Add** to open the **Project Selection** dialog.
4. From the list of available library projects, select a project and click **OK**.
5. When the dialog closes, click **Apply** in the **Properties** window.
6. Click **OK** to close the **Properties** window.
7. As soon as the Properties dialog closes, Eclipse rebuilds the project, including the contents of the library project.
8. The figure below shows the Properties dialog that lets you add library references and move them up and down in priority.

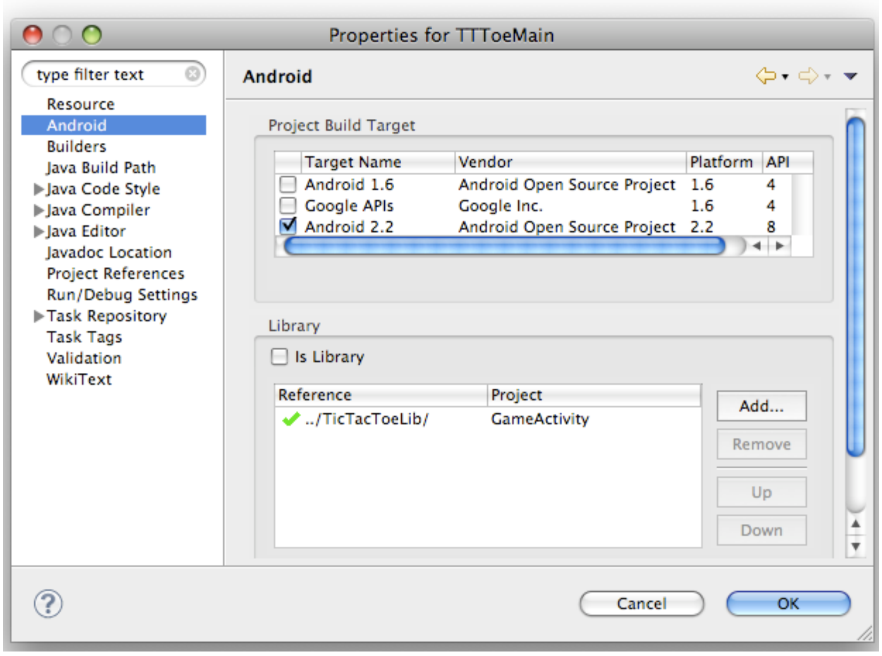


Figure 5-3. Adding a reference to a library project in the properties of an application project.

If you are adding references to multiple libraries, note that you can set their relative priority (and merge order) by selecting a library and using the **Up** and **Down** controls. The tools merge the referenced libraries with your application starting from lowest priority (bottom of the list) to highest (top of the list). If more than one library defines the same resource ID, the tools select the resource from the library with higher priority. The application itself has highest priority and its resources are always used in preference to identical resource IDs defined in libraries.

Declaring library components in the the manifest file

In the manifest file of the application project, you must add declarations of all components that the application will use that are imported from a library project. For example, you must declare any <activity>, <service>, <receiver>, <provider>, and so on, as well as <permission>, <uses-library>, and similar elements.
Declarations should reference the library components by their fully-qualified package names, where appropriate.

For example, the TicTacToeMain example application declares the library Activity GameActivity like this:

```
<manifest>
  ...
  <application>
    ...
    <activity
android:name="com.example.android.tictactoe.library.GameActivity"
/>
    ...
  </application>
</manifest>
```

Development considerations

As you develop your library project and dependent applications, keep the points listed below in mind.

Resource conflicts

Since the tools merge the resources of a library project with those of a dependent application project, a given resource ID might be defined in both projects. In this case, the tools select the resource from the application, or the library with highest priority, and discard the other resource. As you develop your applications, be aware that common resource IDs are likely to be defined in more than one project and will be merged, with the resource from the application or highest-priority library taking precedence.

Using prefixes to avoid resource conflicts

To avoid resource conflicts for common resource IDs, consider using a prefix or other consistent naming scheme that is unique to the project (or is unique across all projects).

No export of library project to JAR

A library cannot be distributed as a binary file (such as a jar file). This is because the library project is compiled by the main project to use the correct resource IDs.

A library project can include a JAR library

You can develop a library project that itself includes a JAR library, however you need to manually edit the dependent application project's build path and add a path to the JAR file.

A library project can depend on an external JAR library

You can develop a library project that depends on an external library (for example, the Maps external library). In this case, the dependent application must build against a target that includes the external library (for example, the Google APIs Add-On). Note also that both the library project and the dependent application must declare the external library their manifest files, in a <uses-library> element.

Library project can not include raw assets

The tools do not support the use of raw asset files in a library project. Any asset resources used by an application must be stored in the assets/ directory of the application project itself.

Targeting different Android platform versions in library project and application project

A library is compiled as part of the dependent application project, so the API used in the library project must be compatible with the version of the Android library used to compile the application project. In general, the library project should use an API level that is the same as — or lower than — that used by the application. If the library project uses an API level that is higher than that of the application, the application project will fail to compile. It is perfectly acceptable to have a library that uses the Android 1.5 API (API level 3) and that is used in an Android 1.6 (API level 4) or Android 2.1 (API level 7) project, for instance.

No restriction on library package name

There is no requirement for the package name of a library to be the same as that of applications that use it.

Multiple R classes in gen/ folder of application project

When you build the dependent application project, the code of any libraries is compiled and merged to the application project. Each library has its own R class, named according to the library's package name. The R class generated from the resources of the main project and of the library is created in all the packages that are needed including the main project's package and the libraries' packages.

Testing a library project

There are two recommended ways of setting up testing on code and resources in a library project:

1. You can set up a test project that instruments an application project that depends on the library project. You can then add tests to the project for library-specific features.

2. You can set up a set up a standard application project that depends on the library and put the instrumentation in that project. This lets you create a self-contained project that contains both the tests/instrumentations and the code to test.

Library project storage location

There are no specific requirements on where you should store a library project, relative to a dependent application project, as long as the application project can reference the library project by a relative link. You can place the library project What is important is that the main project can reference the library project through a relative link.

Migrating library projects to ADT 0.9.8

This section provides information about how to migrate a library project created with ADT 0.9.7 to ADT 0.9.8 (or higher). The migration is needed only if you are developing in Eclipse with ADT and assumes that you have also upgraded to SDK Tools r7 (or higher).

The way that ADT handles library projects has changed between ADT 0.9.7 and ADT 0.9.8. Specifically, in ADT 0.9.7, the src/ source folder of the library was linked into the dependent application project as a folder that had the same name as the library project. This worked because of two restrictions on the library projects:

- The library was only able to contain a single source folder (excluding the special gen/ source folder), and
- The source folder was required to have the name src/ and be stored at the root of the project.

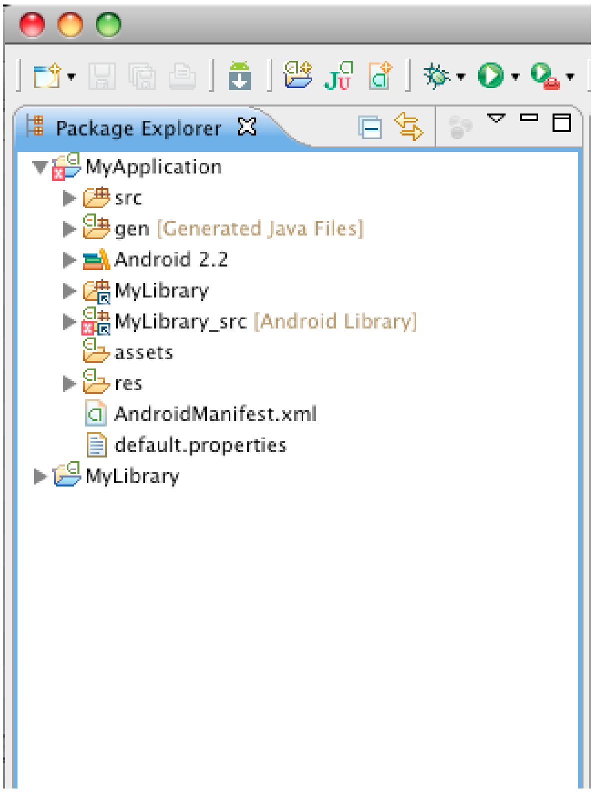
In ADT 0.9.8, both of those restrictions were removed. A library project can have as many source folders as needed and each can have any name. Additionally, a library project can store source folders in any location of the project. For example, you could store sources in a src/java/ directory. In order to support this, the name of the linked source folders in the main project are now called <*library-name*>_<*folder-name*> For example: MyLibrary_src/ or MyLibrary_src_java/.

Additionally, the linking process now flags those folders in order for ADT to recognize that it created them. This will allow ADT to automatically migrate the project to new versions of ADT, should they contain changes to the handling of library projects. ADT 0.9.7 did not flag the linked source folders, so ADT 0.9.8 cannot be sure whether the old linked folders can be removed safely. After upgrading ADT to 0.9.8, you

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.
will need to remove the old linked folders manually in a simple two-step process, as described below.

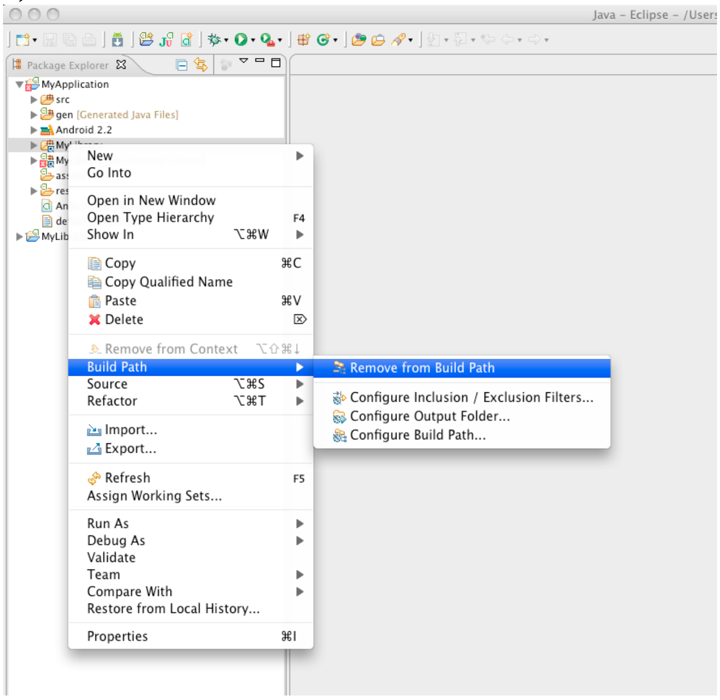
Before you begin, make sure to create a backup copy of your application or save the latest version to your code version control system. This ensures that you will be able to easily revert the migration changes in case there is a problem in your environment.

When you first upgrade to ADT 0.9.8, your main project will look as shown below, with two linked folders (in this example, MyLibrary and MyLibrary_src — both of which link to MyLibrary/src. Eclipse shows an error on one of them because they are duplicate links to a single class.

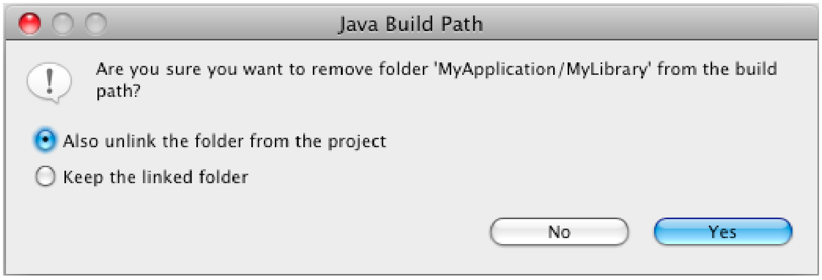


To fix the error, remove the linked folder that *does not* contain the _src suffix.

1. Right click the folder that you want to remove (in this case, the MyLibrary folder) and choose **Build Path** > **Remove from Build Path**, as shown below.



2. Next, When asked about unlinking the folder from the project, select **Yes**, as shown below.



This should resolve the error and migrate your library project to the new ADT environment.

Eclipse Tips

Executing arbitrary Java expressions in Eclipse

You can execute arbitrary code when paused at a breakpoint in Eclipse. For example, when in a function with a String argument called "zip", you can get information about packages and call class methods. You can also invoke arbitrary static methods: for example, entering android.os.Debug.startMethodTracing() will start dmTrace.

Open a code execution window, select **Window** > **Show View** > **Display** from the main menu to open the Display window, a simple text editor. Type your expression, highlight the text, and click the 'J' icon (or CTRL + SHIFT + D) to run your code. The code runs in the

context of the selected thread, which must be stopped at a breakpoint or single-step point. (If you suspend the thread manually, you have to single-step once; this doesn't work if the thread is in Object.wait().)

If you are currently paused on a breakpoint, you can simply highlight and execute a piece of source code by pressing CTRL + SHIFT + D. You can highlight a block of text within the same scope by pressing ALT +SHIFT + UP ARROW to select larger and larger enclosing blocks, or DOWN ARROW to select smaller blocks.

Here are a few sample inputs and responses in Eclipse using the Display window.

Input	**Response**
zip	(java.lang.String) /work/device/out/linux-x86-debug/android/app/android_sdk.zip
zip.endsWith(".zip")	(boolean) true
zip.endsWith(".jar")	(boolean) false

5.5.4 Android AVD

Creating an AVD

An Android Virtual Device (AVD) is a device configuration for the emulator that allows you to model real world devices. In order to run an instance of the emulator, you must create an AVD.

To create an AVD from Eclipse:

1. Select **Window > Android SDK and AVD Manager**, or click the Android SDK and AVD Manager icon in the Eclipse toolbar.
2. In the *Virtual Devices* panel, you'll see a list of existing AVDs. Click **New** to create a new AVD.
3. Fill in the details for the AVD.
   Give it a name, a platform target, an SD card size, and a skin (HVGA is default).
   > **Note:** Be sure to define a target for your AVD that satisfies your application's Build Target (the AVD platform target must have an API Level equal to or greater than the API Level that your application compiles against).
4. Click **Create AVD**.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Your AVD is now ready and you can either close the SDK and AVD Manager, create more AVDs, or launch an emulator with the AVD by selecting a device and clicking **Start**.

5.6 Project Framework

A framework is the base of your future application. Its usage greatly simplifies the whole development process. Instead of writing an application from scratch and dealing with large portions of code to make your application work on different platforms – you use a framework. Here’s a list of framework for mobile app development:

5.6.1 Apple IOS

iOS, which was previously called iPhone OS, is a mobile operating system developed by Apple Inc. Its first release was in 2007, which included iPhone and iPod Touch. iPad (1st Generation) was released in April 2010 and iPad Mini was released in November 2012.

The iOS devices get evolved quite frequently and from experience, we find that at least one version of iPhone and iPad is launched every year. Now, we have iphone5 launched which has its predecessors starting from iPhone, iPhone 3gs, iPhone 4, iPhone 4s. Similarly, iPad has evolved from iPad (1^{st} Generation) to iPad (4^{th} Generation) and an additional iPad Mini version.

The iOS SDK has evolved from 1.0 to 6.0. iOS 6.0, the latest SDK is the only officially supported version in Xcode 4.5 and higher. We have a rich Apple documentation and we can find which methods and libraries can be used based on our deployment target. In the current version of Xcode, we’ll be able to choose between deployment targets of iOS 4.3, 5.0 and 6.0.

The power of iOS can be felt with some of the following features provided as a part of the device.

- Maps
- Siri
- Facebook and Twitter
- Multi-Touch
- Accelerometer
- GPS
- High end processor

- Camera
- Safari
- Powerful APIs
- Game center
- In-App Purchase
- Reminders
- Wide Range of gestures

The number of users using iPhone/iPad has increased a great deal. This creates the opportunity for developers to make money by creating applications for iPhone and iPad the Apple's App Store.

For some one new to iOS, Apple has designed an application store where the user can buy apps developed for their iOS devices. A developer can create both free and paid apps to App Store. To develop applications and distribute to the store, the developer will require to register with iOS developer program which costs $99 a year and a Mac with Mountain Lion or higher for its development with latest Xcode.

Registering as an Apple Developer

An Apple ID is most necessary if you are having any Apple device and being a developer, you definitely need it. It's free and hence, no issues in having one. The benefits of having an Apple account are as follows −

- Access to development tools.
- Worldwide Developers Conference (WWDC) videos.
- Can join iOS developer program teams when invited.

To register an Apple account, follow the steps given below −

Step 1 − Click the link https://developer.apple.com/programs/register/ and select "Create Apple ID"



Step 2 − Provide the necessary information, which is self explanatory as given in the page.

Step 3 − Verify your account with your email verification and the account becomes active.

Step 4 − Now you will be able to download the developer tools like Xcode, which is packaged with iOS simulator and iOS SDK, and other developer resources.

Apple iOS Developer Program

The first question that would arise to a new developer is – Why should I register for an iOS developer program? The answer is quite simple; Apple always focuses on providing quality applications to its user. If there was no registration fee, there could be a possibility of junk apps being uploaded that could cause problems for the app review team of Apple.

The benefits of joining the iOS developer program are as follows −

- Run the apps you develop on the real iOS device.
- Distribute the apps to the app store.
- Get access to the developer previews.

The steps to join the iOS developer program are as follows −

Step 1 − To register, use the link – (https://developer.apple.com/programs/ios/).



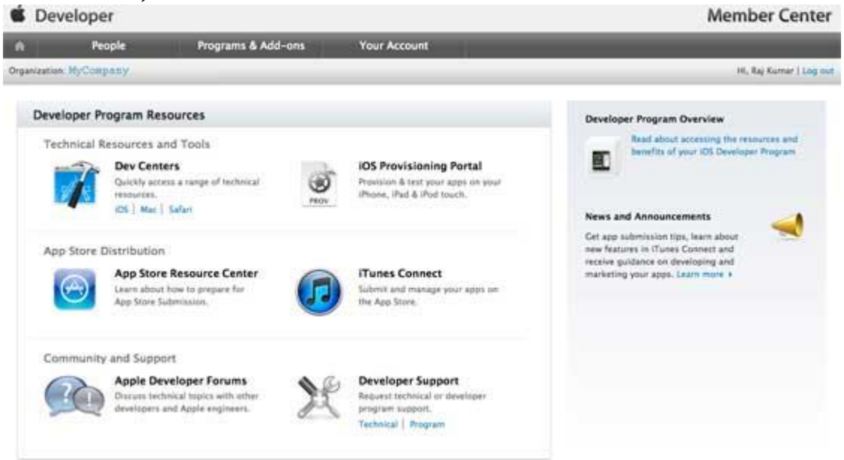
Step 2 − Click on Enroll Now in the page that is displayed.

Step 3 − You can either sign in to your existing apple account (if you have one) or create a new Apple ID.

Step 4 − Thereafter, you have to select between Individual and Company accounts. Use company account if there will be more than one developer in your team. In individual account, you can't add members.

Step 5 − After entering the personal information (for those who newly registers), you can purchase and activate the program by paying with the help of your credit card (only accepted mode of payment).

Step 6 − Now you will get access to developer resources by selecting the member center option in the page.

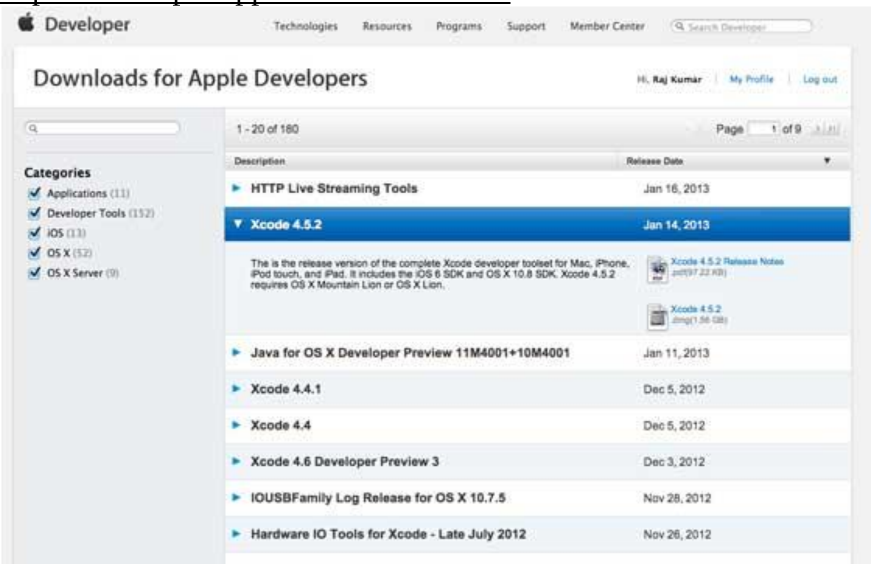


Step 7 − Here you will be able to do the following −

- Create provisioning profiles.
- Manage your team and devices.
- Managing application to app store through iTunes Connect.
- Get forum and technical support.

iOS - Xcode Installation

Step 1 − Download the latest version of Xcode from https://developer.apple.com/downloads/



Step 2 − Double click the Xcode dmg file.

Step 3 − You will find a device mounted and opened.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

Step 4 − There will be two items in the window that's displayed namely, Xcode application and the Application folder's shortcut.

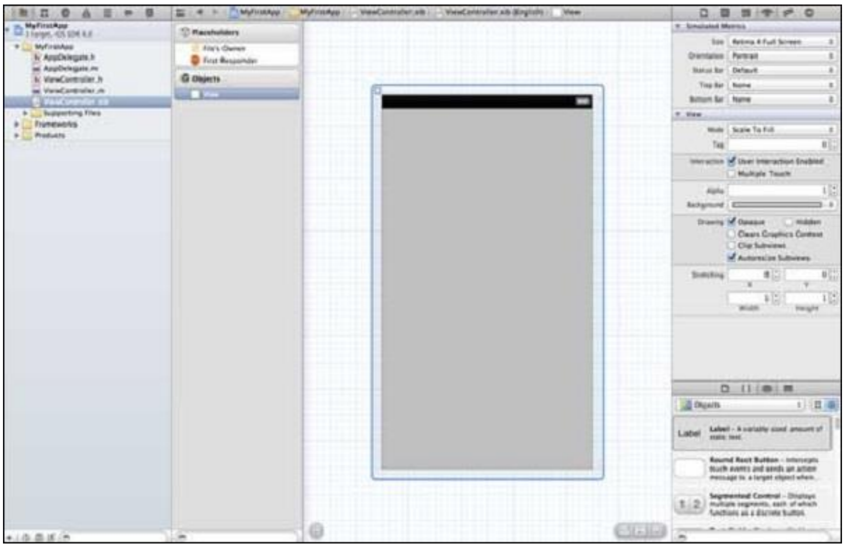
Step 5 − Drag the Xcode to application and it will be copied to your applications.

Step 6 − Now Xcode will be available as a part of other applications from which you can select and run.

You also have another option of downloading Xcode from the Mac App store and then install following the step-by-step procedure given on the screen.

Interface Builder

Interface builder is the tool that enables easy creation of UI interface. You have a rich set of UI elements that is developed for use. You just have to drag and drop into your UI view. We'll learn about adding UI elements, creating outlets and actions for the UI elements in the upcoming pages.



You have objects library at the right bottom that consists the entire necessary UI element. The user interface is often referred as **xibs**, which is its file extension. Each of the xibs is linked to a corresponding view controller.

iOS Simulator

An iOS simulator actually consists of two types of devices, namely iPhone and iPad with their different versions. iPhone versions include iPhone (normal), iPhone Retina, iPhone 5. iPad has iPad and iPad Retina. A screenshot of an iPhone simulator is displayed below.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.



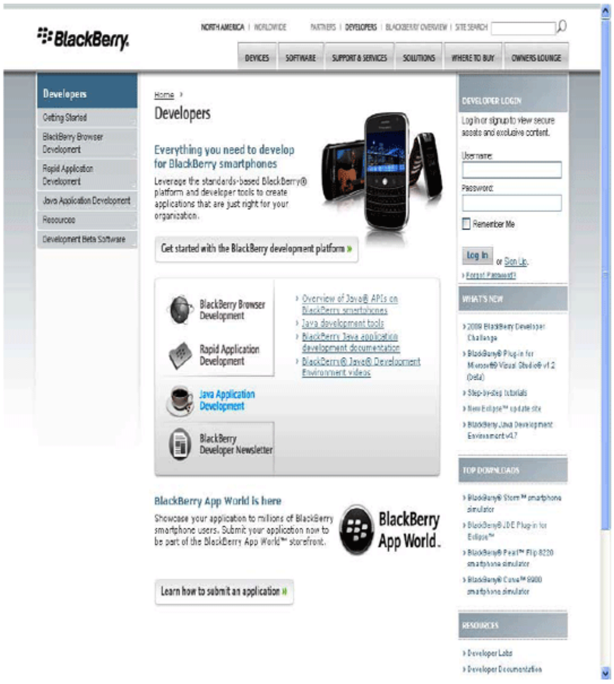
You can simulate location in an iOS simulator for playing around with latitude and longitude effects of the app. You can also simulate memory warning and in-call status in the simulator. You can use the simulator for most purposes, however you cannot test device features like accelerometer. So, you might always need an iOS device to test all the scenarios of an application thoroughly

5.6.2 RIM Blackberry

Blackberry is a brand of mobile phones developed by a Canadian telecommunication company called Research in Motion (RIM). Most of these Blackberry phones are smart phones and are popularly known for their ability to send and receive instant messages and push emails and maintaining a high standard of security by encryption.

The BlackBerry Developer Zone

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.



While you’re looking at this page, you might as well sign up for a developer account – it’s free and quick, and you’ll need a login to download the developer tools. RIM does offer higher-level paid developer programs with additional support and other benefits, but you can develop and distribute applications with the free account.

Installing the Development Environment

There are two BlackBerry development environments produced by RIM. The BlackBerry Java Development Environment (JDE), and the BlackBerry JDE Plug-in for Eclipse. Both are very functional and have been used by developers to produce professional applications.

The JDE has been around longer and is a bit more mature, but almost everything possible with the JDE can also be accomplished with the Eclipse Plug-in. The Eclipse Plug-in leverages the entire Eclipse development platform, which includes a world-class source code editor and a lot of third-party plug-ins. Ultimately, the choice is a matter of personal preference. We’ll explore both in the next chapter, so you’ll get a better idea of what the real-world differences are. There are no issues with installing both the JDE and the JDE Plug-in for Eclipse on the same computer, so if you’re interested in exploring both and don’t mind the extra time and effort, feel free to follow through the install instructions for both later in this.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

After deciding between the JDE and the JDE Plug-in, you'll need to decide on a JDE version. Each version of the JDE (or each version of the component pack for the Eclipse Plug-in) corresponds to a major version of the BlackBerry operating system (OS). BlackBerry does a good job of keeping their OS backward compatible, so something developed for OS 4.2 generally will work the same on OS 4.3 and higher.

However, you may want to use some features that are only available in a later OS. A safe minimum is 4.2, which covers all trackball devices and later and is the minimum version supported by BlackBerry App World. The one exception to all of this is the touch screen BlackBerry Storm, which runs OS 4.7 and can be temperamental with applications built using older versions of the JDE. You can run applications compiled with versions of the JDE earlier than JDE v4.7 on the Storm, and they will work. However, by default, they'll be run in Compatibility Mode, meaning the user experience won't be ideal.

To avoid Compatibility Mode, you must compile your application with JDE v4.7 or higher. In many cases, you can just recompile the same source code. The bottom line is that if you're planning on targeting the Storm, you should be sure to get the JDE or JDE Plug-in v4.7 in addition to any other versions. Before installing the BlackBerry development tools, you'll need to install the Java SE. The version or versions you will have to install depends on the version of the BlackBerry platform you want to target.

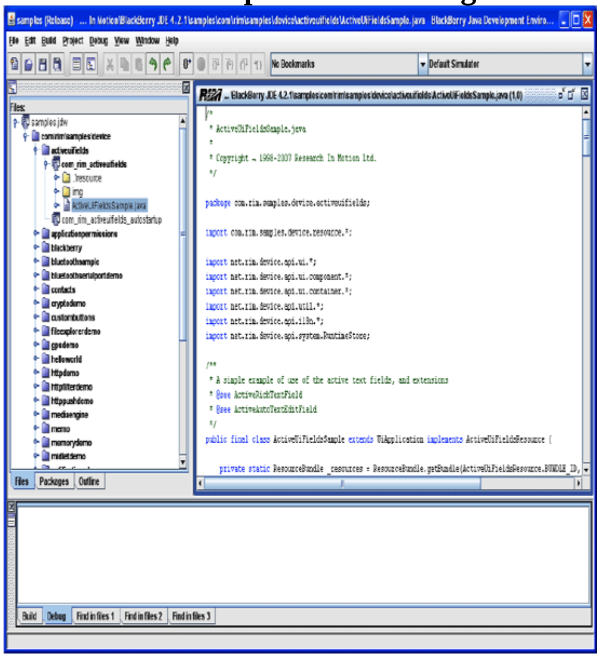
For most developers, downloading Java SE JDK v6.0 is a good choice – it will let you develop for BlackBerry Device Software version 4.2 and later, which covers all BlackBerry devices introduced in the last three years or so. More specific information is available on the Developer Zone

Installing the BlackBerry JDE

The JDE is a fully integrated stand-alone environment, so if you have the appropriate version of the Java Development Kit (JDK) installed, you just need to download the appropriate version of the JDE installer and run it. Everything you need for BlackBerry development is included in the JDE – from writing code using the built-in editor, to debugging using the array of BlackBerry device simulators available, to building and signing your application for deployment onto real devices.This shows the BlackBerry JDE as it will appear after being launched for the first time.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

The BlackBerry JDE with the excellent (included) Samples workspace loaded, and a Java source file opened for editing



5.6.3 Samsung Bada

Its Mobile operating system designed for Smart Phones by Samsung Electronics. The name ,bada which means ocean in Korean, was chosen to convey the limitless variety of potential applications which can be created using the new platform. It is one of the most developer-friendly environments available, particularly in the area of applications using Web services. Samsung bada also represents the fresh challenges and opportunities available to developers, as well as the entertainment which consumers will enjoy once the new platform is open.

Behind the bar

Samsung did a survey of consumers mobile software demands. Results were found to be extremely encouraging for mobile application developers. This was identified as a great opportunity for new development. Even with the current feature set provided by the smart phones available in market, 42% of the current smart phone users surveyed would pay to download applications, if they could. This illustrates the scale of the potential revenues available for developers by extending the market of downloadable application for smart phones.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

As of now, Apple, RIM (Research In Motion – BlackBerry) and Google dominate the Smart Phone OS and Mobile Application Market. Samsung which is the second largest manufacturer of mobile phones also want the pie of this lucrative high end mobile application market. Recent statistics available show that smart phones will account for 16% of cell phone market this year, which makes more room for Samsung in this sector.

The aim of Bada platform is to make smart phone features accessible to everyone, so that developers can reach larger audiences and the variety and creativity of apps can be enjoyed by many. They want consumers to have a fun and diverse mobile experience that really adds value to their lives, by providing them with high-quality application and mobile services.

Key Features

Extensible core functions

A call dialer, messaging and address book, which Bada applications can freely use. Bada will give developer the chance to access phones' accelerometers, tilt, weather, proximity and activity sensors, so they can build apps that respond to tilting the phone.

Smart phones for everyone

More and more people want the rich and connected application-experiences that are currently only available for smart phone consumers. Samsung has developed bada to make these exclusive smart phone experiences available to everyone

Tools

The new UI tool includes the ability to embed the AdobeFlashPlayer and WebKit Internet Browser directly into native Bada applications. Also Bada map Control can be used for mapping in applications. Also Eclipse and GNU tool-chain can be used as IDE for development purpose.

Feature-rich developer platform

The new OS will deliver simple, instinctive, and innovative visual design using Next Generation UI Framework. Bada supports motion sensor and face detection. Also it provides mechanism to develop sensor based and context-aware applications.

Service Oriented Features

Developers can create service-centric applications like social networking applications for managing user profiles, location applications for mapping etc.

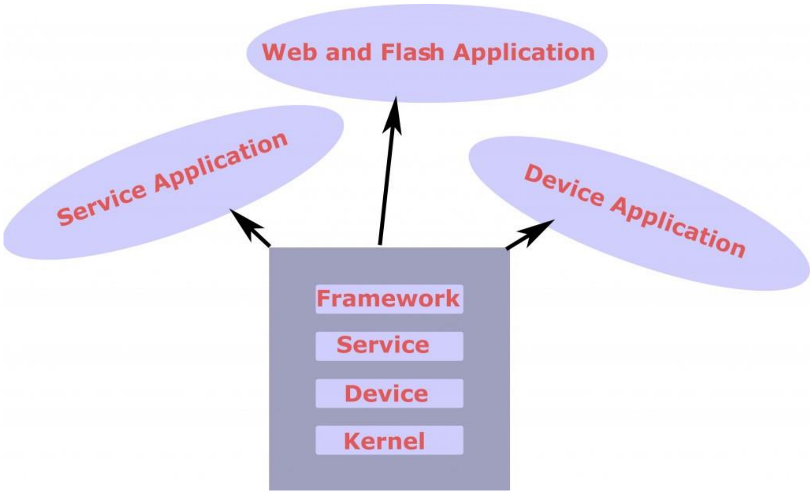
Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

And all the bada handsets will have some common features:

1. 3G / Wi-Fi
2. GPS / Motion Sensor
3. WGVGA/WVGA screen
4. Multipoint touch

Bada supports third party services like Twitter and Facebook through its APIs.

Architecture



Bada Architecture Stack

The Architecture is made up of four layers

I. Kernel Layer

This is based on real-time Operating System or Linux kernel based depending on hardware configuration. Speculation is that it can be linux based system

II.Device Layer

This provides the core functions of a device platform that are provided by OS, such as graphics and multimedia, and communication components.

III. Service Layer

Service-centric functions that are provided by application engine like messaging and contact management.

IV. Framework Layer

Open API framework that consist of an application framework and functions exported by underlying layers. One can create innovative application on Bada using C++

Bada / Android

Android mobile phone industry has gradually becoming center of attraction. Since Android supports java which has big community support. HTC, Motorola, Sony Ericsson has now adopted Android platform in their newly launched handsets.

Where as Bada is proprietary of Samsung so the Bada will be inbuilt in Samsung smart phones but there are less chances of adaptation of Bada by other Mobile Manufacturers as it is direct threat to them. However Samsung is the second largest mobile manufacturer. This will increase the chances of success of Bada. Also the Games developers CAPCOM, EA Mobile, and Gameloft are supporting Bada.

To Get Started

To get started with Bada application development install SDK and IDE. After registration one needs to become partner to download SDK and IDE.

5.6.4 Nokia Symbian

Symbian's predecessor, Symbian OS, was developed by Symbiant Ltd, a partnership among PDA and smartphone manufacturers Nokia, Ericsson, Motorola and Psion. Nokia expressed interest in acquiring the entire company and the acquisition was completed at the end of 2008.

Targeted for smartphones, Symbian is designed to thrive in low-power battery-based devices as well as ROM-based systems. Its kernel, known as EKA2 (EPOC Kernel Architecture 2), features preemptive multithreading and full memory protection. This kernel already contains a scheduler, a memory management system and device drivers.

Applications for Symbian are normally written in C++ (using Qt) or Symbian C++. However, applications written in Python, Java ME, Flash Lite, Ruby and .NET can also run. These applications may then be installed on the device using OTA (over-the-air), a mobile to PC data cable connection, Bluetooth or a memory card.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

5.6.5 Microsoft Windows Phone

Developing mobile applications for Microsoft's Windows Phone platform is a straightforward process with many tools available to developers. In fact, the mobile team at Microsoft did a great job with the Windows Phone platform by taking a completely unique approach in several aspects.

Prerequisites

We're going to create a basic Windows Phone 8 application using C#. You need to have Visual Studio installed as well as the Windows Phone 8 SDK. If you don't have Visual Studio installed, then I recommend you install Visual Studio Express 2012, which you can download from Microsoft's Download Center. This will install the necessary software and tools for Windows Phone 8 development.

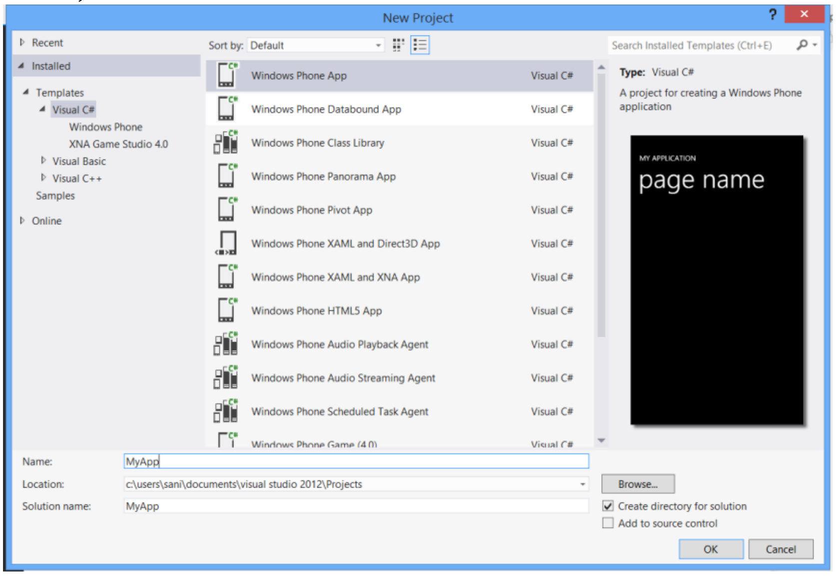
By installing Visual Studio Express 2012 for Windows Phone 8, you install the following applications and tools:

- Visual Studio Express 2012
- .Net Framework
- Windows Phone 8 SDK
- Blend For Visual Studio 2012

With the development tools installed, it's time to start creating your very first Windows Phone 8 application.

2. Your First Application

To create a new Windows Phone project, launch Visual Studio Express 2012 and select **New Project > Windows Phone App** from the **File** menu. Give the project a name, specify a location to save the project to, and click **OK.**

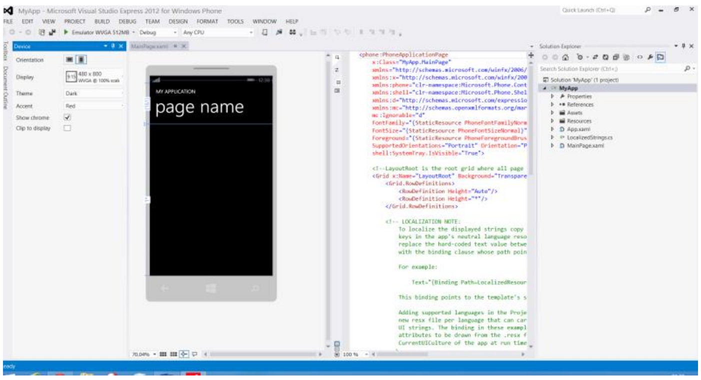


Make sure to select the **Visual C#** template from the list of **Templates** on the left. Visual Studio will also ask you about the version you want to target. We'll be targeting version 8.0.

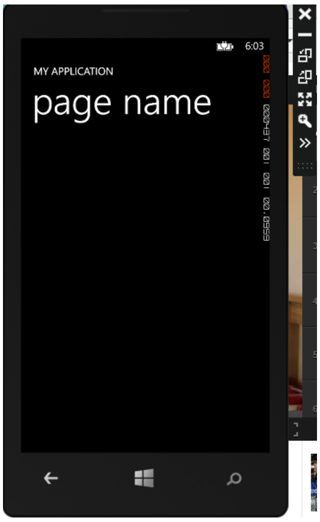
Advertisement

3. Launch Your Application

We now have a very basic Windows Phone project to work with. Let's launch the application to see what it actually looks like on the Windows Phone emulator.



To run your application in the Windows Phone emulator, click the green play button at the top left of the window. This will launch the emulator, install your application, and launch it in the emulator.



The result is a screen similar to the one shown below. You can choose between several emulators in Visual Studio. Feel free to play around with the other emulators, but don't choose the Windows Phone 7 emulator as our project targets Windows Phone 8.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

5.7 Check your Progress Questions

1. The **______** operating system is used as the base of the Android stack.

2. When developing for the Android OS, Java bytecode is compiled into what?

a) Java source code b) Dalvik application code

c) Dalvik byte code d) C source code

3. iOS stands for ?

a)Internetwork Operating System b)iPhone Operating System

c)Internet Operating System d)None of Them

4. Which one of the following is not an application development framework

a)SDK bB)Eclipse

c)CASE d)Android AVD

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

5.8 Answers to check your progress questions.

1. Linux
2. c) Dalvik byte code
3. b)iPhone Operating System
4. c)CASE

5.9 Summary

Smart Phones has changed the life of everyone. Along with other features, an App in Smart Phones allows to do almost everything, from playing games to do business. Before your apps are ready for their official release to the public, they should be thoroughly tested and verified in as many devices and software configurations as possible. With the gigantic number of Android devices available out there, this requirement for hardware testing is almost unrealistic, especially for independent developers with limited financial resources. Therefore, using the SDK and AVD managers to create and emulate some of the most popular devices becomes one of the most affordable approaches.

SDK Manager also will offer you the opportunity to preview the most recent features before they even exist on any physical devices. AVD Manager then allows you to configure virtual devices to resemble the software and hardware combinations for those devices already in the market. While the smart phone contains advanced features which is based on the operating system that allows to run software applications.

The most common mobile Operating Systems (OS) used by modern smartphones include Google's Android, Apple's iOS, Nokia's Symbian, RIM's BlackBerry OS, Samsung's Bada, Microsoft's Windows Phone. Such operating systems can be installed on different phone models, and they can receive multiple OS software updates over its lifetime.

5.10 Key words

Google Android Application Development, SDK, Eclipse, Emulator, Android AVD, Apple IOS, RIM Blackberry, Samsung Bada, Nokia Symbian, Microsoft Windows Phone.

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

5.11 Self Assessment Questions and Answers

Short Answer Questions

1. What is Google Android?
2. Define SDK.
3. Write the features of Android.
4. What is meant by Apple IOS.
5. Write about RIM Blackberry.
6. Outline the various challenges Android Application Development framework.

Long Answer Questions

1. Elaborate the use of Software development framework (SDK).
2. Explain the following application development framework
    a. SDK
    b. Eclipse
    c. Emulator
    d. Android AVD
3. Discuss Google Android Application Development in detail
4. Explain the following Project Framework
    a. Apple IOS
    b. RIM Blackberry
    c. Samsung Bada
    d. Nokia Symbian
    e. Microsoft Windows Phone
5. Explain Nokia Symbian in detail.
6. Discuss Microsoft Windows Phone.

5.12 Further readings

1. Mobile Design and Development by Brian Fling, O'Reilly Media, Inc 2009

2. J2ME: The Complete Reference, James Keogh, Tata McGrawHill 2003

3. H. Lashkari, M. Moradhaseli, Mobile operating systems and programming: mobile communications, VDM Verlag Dr. Muller, 2011

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

MODEL QUESTION PAPER

PART A – (10 x 2 = 20)

Answer ALL questions

1. What do we mean by mobile ecosystems?
2. List out some platforms to build mobile application.
3. Define short message service (SMS).
4. Why Informative Apps are very important in learning?
5. What is Information Architecture?
6. Mention some ways of mobile prototype.
7. What is the difference between J2ME and J2SE?
8. Write the layers of J2ME architecture.
9. Write the features of Android.
10. What is meant by Apple IOS.

PART B – (5 x 5 = 25)

Answer ALL questions

11. (a) Explain The Mobile Ecosystem with the following concepts
    a) Operators
    b) Networks
    c) Devices
    d) Platforms
    e) Services

Or

(b) Discuss the application frameworks of the mobile ecosystem in detail.

12. (a) Explain the following concepts in detail
    a) Native apps
    b) Hybrid apps
    c) Web apps
    d) Utility apps

Or

(b) Write the working principles of Location Based Services(LBS).

13. (a) List and explain elements of mobile design.

Or

(b) Explain the following Mobile Information Architecture.
    a) Sitemaps
    b) Click Streams
    c) Wireframes
    d) Prototyping
    e) Architecture

Author: Dr.P.Thiyagarajan, Dept. of Computer Science, Central University of Tamil Nadu, Thiruvarur – 610 005.

14. (a) Explain in detail about ‘Hello world’ application using wireless toolkit.

Or

(b) What is midlet suite? Explain the MIDlet lifecycle in detail.

15. (a) Explain Nokia Symbian in detail.

Or

(b) Discuss Microsoft Windows Phone.

PART C – (3 x 10 = 30)

Answer any THREE questions

16. Draw the mobile ecosystem architecture and explain each component clearly.

17. Explain the following concepts in detail

a) Enterprise app
b) Informative App
c) Utility App
d) Mobile web App

18. Discuss mobile design tools and elements of mobile design.

19. Draw and explain J2ME architecture with development and run time environment.

20. Explain the following application development framework

a) SDK
b) Eclipse
c) Emulator
d) Android AVD